

SYSTEM DATA MODEL (SDM) SOURCE CODE

James C. Lyke and Josette Calixte-Rosengren

23 Aug 2012

Technical Paper

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.



**AIR FORCE RESEARCH LABORATORY
Space Vehicles Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776**

DTIC COPY NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 377 ABW Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RV-PS-TP-2012-0062 HAS BEEN REVIEWED AND IS APPROVED FOR
PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT

//SIGNED//
JAMES LYKE
Program Manager

//SIGNED//
KEN HUNT
Tech Advisor, Space Electronics Protection Branch

//SIGNED//
KEN D. BOLE, Lt Col, USAF
Deputy Chief, Spacecraft Technology Division
Space Vehicles Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YY) 23-08-2012		2. REPORT TYPE Technical Paper		3. DATES COVERED (From - To) 01 Jan 2008 – 31 Dec 2009	
4. TITLE AND SUBTITLE System Data Model (SDM) Source Code				5a. CONTRACT NUMBER TRDP-US-SW-AF-09-0002	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) James C. Lyke and Josette Calixte-Rosengren				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Space Vehicles Directorate 3550 Aberdeen Ave SE Kirtland AFB, NM 87117-5776				8. PERFORMING ORGANIZATION REPORT NUMBER AFRL-RV-PS-TP-2012-0062	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RVSE	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited. (377ABW-2012-1217, dtd 14 Sep 12)					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT This document provides the source code of the System Data Model (SDM), formerly "Satellite Data Model" to the public domain.					
15. SUBJECT TERMS Plug-and-Play architecture, software					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Unlimited	18. NUMBER OF PAGES 3422	19a. NAME OF RESPONSIBLE PERSON James C. Lyke
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (include area code)

(This page intentionally left blank)

Table of Contents

Section	Page
Introduction.....	1
Appendix.....	2
Bibliography	3408
List of Abbreviations, Acronyms, and Symbols.....	3414

(This page intentionally left blank)

Introduction

This document conveys the source code for the System Data Model (SDM, formerly “Satellite Data Model”), which was developed from 2004-2009 (through a variety of contract and in-house efforts) as part of the Air Force Research Laboratory (AFRL) Space Plug-and-play Architecture (SPA) research program. The software was developed as a middleware to explore plug-and-play concepts and to develop a better understanding of how to make join and discovery mechanisms operate in embedded systems.

This document only presents the source code of SDM in its entirety as a retired work to the public domain for study and follow-on research activities. The theory and operation of SDM will not be explained here, for a better understanding of SDM and SPA, the reader is referred to the bibliography following the Appendix.

The source code in its entirety is presented in the sole Appendix. AFRL does not endorse, warranty, or otherwise restrict the use of this code.

APPENDIX

System Data Model Source Code

File listing of SDM source tree library (Table of Contents)

sdm/Makefile.defs
sdm/Makefile.defs.uclinux
sdm/Makefile.common
sdm/Makefile.uclinux
sdm/Makefile
sdm/Spa1Manager/Spa1Translator.h
sdm/Spa1Manager/Spa1AsimTable.h
sdm/Spa1Manager/Spa1Msg.cpp
sdm/Spa1Manager/Spa1Msg.h
sdm/Spa1Manager/Spa1Queue.cpp
sdm/Spa1Manager/Spa1Queue.h
sdm/Spa1Manager/Spa1Translator.cpp
sdm/Spa1Manager/Spa1AsimTable.cpp
sdm/Spa1Manager/Makefile
sdm/Spa1Manager/Spa1Manager.cpp
sdm/Spa1Manager/Spa1Asim.h
sdm/Spa1Manager/Spa1Asim.cpp
sdm/Spa1Manager/Spa1Manager.h
sdm/Spa1Manager/Spa1ManagerxTEDS.h
sdm/pm/pm.cpp
sdm/pm/pm_ids.cpp
sdm/pm/HeartbeatTestApp.cpp
sdm/pm/pmxDTEDS.h
sdm/pm/pm_monitor.cpp
sdm/pm/PMProcess.h
sdm/pm/PendingTask.h
sdm/pm/Makefile.uclinux
sdm/pm/Makefile
sdm/pm/.wrmakefile
sdm/pm/pm_main.h
sdm/pm/PMProcessList.cpp
sdm/pm/PendingTask.cpp
sdm/pm/.wrproject
sdm/pm/PMProcessList.h
sdm/pm/PMProcess.cpp
sdm/pm/pm_main.cpp

sdm/pm/pm.h
sdm/pm/pm_ids.h
sdm/pm/.project
sdm/sm/sensor.cpp
sdm/sm/sm.h
sdm/sm/SensorRecord.h
sdm/sm/SensorMonitor.cpp
sdm/sm/sm.cpp
sdm/sm/Makefile
sdm/sm/SensorRecord.cpp
sdm/sm/sensor.h
sdm/sm/sm_monitor.cpp
sdm/sm/SensorMonitor.h
sdm/sm/sm_monitor_win32.cpp
sdm/tm/TmXtedsDefs.h
sdm/tm/pm_record.h
sdm/tm/pm_record_list.h
sdm/tm/tm.h
sdm/tm/tm_monitor_win32.cpp
sdm/tm/tm_monitor.cpp
sdm/tm/pm_record_list.cpp
sdm/tm/Makefile.uclinux
sdm/tm/Makefile
sdm/tm/.wrmakefile
sdm/tm/tasklist.h
sdm/tm/pm_record.cpp
sdm/tm/tm.cpp
sdm/tm/tasklist.cpp
sdm/tm/.wrproject
sdm/tm/SdmTaskList.config
sdm/tm/HandlerArguments.h
sdm/tm/task.cpp
sdm/tm/task.h
sdm/tm/.project
sdm/asim_test/asim_test.cpp
sdm/asim_test/asim_test_gui.cpp
sdm/asim_test/Makefile
sdm/dm/SubscriptionList.cpp
sdm/dm/DMUtils.h
sdm/dm/DMxTEDS.h
sdm/dm/DM.cpp

sdm/dm/xTEDSSegmentBuilder.cpp
sdm/dm/backupDMList.cpp
sdm/dm/ProviderSubscription.cpp
sdm/dm/xTEDSParameters.cpp
sdm/dm/xTEDSLibraryList.cpp
sdm/dm/xTEDSLibrary.cpp
sdm/dm/Parse.cpp
sdm/dm/Makefile.uclinux
sdm/dm/xTEDSLibraryList.h
sdm/dm/Makefile
sdm/dm/.wrmakefile
sdm/dm/Subscription.cpp
sdm/dm/ProviderSubscriptionList.h
sdm/dm/dm_monitor.cpp
sdm/dm/backupDMList.h
sdm/dm/DMUtils.cpp
sdm/dm/ProviderSubscriptionList.cpp
sdm/dm/SubscriptionList.h
sdm/dm/.wrproject
sdm/dm/xTEDSLibrary.h
sdm/dm/DM.h
sdm/dm/Subscription.h
sdm/dm/xTEDSParameters.h
sdm/dm/xTEDSSegmentBuilder.h
sdm/dm/Parse.h
sdm/dm/ProviderSubscription.h
sdm/dm/.project
sdm/VxWorks/includes/endian.h
sdm/VxWorks/includes/byteswap.h
sdm/VxWorks/bin/regex/libRegex.a
sdm/VxWorks/libRegex/pcre_fullinfo.c
sdm/VxWorks/libRegex/pcre_chartables.c
sdm/VxWorks/libRegex/pcrecpp_internal.h
sdm/VxWorks/libRegex/pcre_maketables.c
sdm/VxWorks/libRegex/pcre_info.c
sdm/VxWorks/libRegex/pcrecpparg.h
sdm/VxWorks/libRegex/pcre_ord2utf8.c
sdm/VxWorks/libRegex/pcre_stringpiece.h
sdm/VxWorks/libRegex/pcre_config.c
sdm/VxWorks/libRegex/pcrecpp.cc
sdm/VxWorks/libRegex/pcreposix.c

sdm/VxWorks/libRegex/pcrecpp.h
sdm/VxWorks/libRegex/pcre_exec.c
sdm/VxWorks/libRegex/pcre.h
sdm/VxWorks/libRegex/pcre_globals.c
sdm/VxWorks/libRegex/pcre_dfa_exec.c
sdm/VxWorks/libRegex/pcre_try_flipped.c
sdm/VxWorks/libRegex/Makefile
sdm/VxWorks/libRegex/.wrmakefile
sdm/VxWorks/libRegex/pcreposix.h
sdm/VxWorks/libRegex/pcre_compile.c
sdm/VxWorks/libRegex/pcre_version.c
sdm/VxWorks/libRegex/pcre_refcount.c
sdm/VxWorks/libRegex/pcre_printint.src
sdm/VxWorks/libRegex/.wrproject
sdm/VxWorks/libRegex/pcre_study.c
sdm/VxWorks/libRegex/pcre_tables.c
sdm/VxWorks/libRegex/pcre_get.c
sdm/VxWorks/libRegex/pcre_internal.h
sdm/VxWorks/libRegex/pcre_xclass.c
sdm/VxWorks/libRegex/pcre_valid_utf8.c
sdm/VxWorks/libRegex/config.h
sdm/VxWorks/libRegex/pcre_newline.c
sdm/VxWorks/libRegex/pcre_ucd.c
sdm/VxWorks/libRegex/ucp.h
sdm/VxWorks/libRegex/.project
sdm/VxWorks/apps/converter/.wrmakefile
sdm/VxWorks/apps/converter/.wrproject
sdm/VxWorks/apps/converter/converter.cpp
sdm/VxWorks/apps/converter/.project
sdm/VxWorks/apps/producer/producer.cpp
sdm/VxWorks/apps/producer/.wrmakefile
sdm/VxWorks/apps/producer/.wrproject
sdm/VxWorks/apps/producer/.project
sdm/VxWorks/apps/consumer/.wrmakefile
sdm/VxWorks/apps/consumer/.wrproject
sdm/VxWorks/apps/consumer/consumer.cpp
sdm/VxWorks/apps/consumer/.project
sdm/app/Makefile
sdm/app/examples/Makefile.uclinux
sdm/app/examples/Makefile
sdm/app/examples/producer.cpp

sdm/app/examples/converter.cpp
sdm/app/examples/consumer.cpp
sdm/app/test/ReqRegTest.cpp
sdm/app/test/MessageLogRemoveAll.cpp
sdm/app/test/ReqxTEDStest.cpp
sdm/app/test/README
sdm/app/test/magnetometer_test.cpp
sdm/app/test/RoboHubTest.cpp
sdm/app/test/SearchSubTest.cpp
sdm/app/test/MessageLogAddAll.cpp
sdm/app/test/TatTest.cpp
sdm/app/test/TMSubsTest.cpp
sdm/app/test/xTEDSPoster.cpp
sdm/app/test/DevicesTest.cpp
sdm/app/test/TCPTester.cpp
sdm/app/test/ModificationSearchSubTest.cpp
sdm/app/test/KillTester.cpp
sdm/app/test/FileServiceTest.cpp
sdm/app/test/TestRegEx.cpp
sdm/app/test/MinimalxTEDSgetIP.cpp
sdm/app/test/Makefile.uclinux
sdm/app/test/Makefile
sdm/app/test/testSDMTat.cpp
sdm/app/test/badReqReg1.cpp
sdm/app/test/ReqReg1.4Test.cpp
sdm/app/test/MessageClassTest.cpp
sdm/app/test/TestDMService.cpp
sdm/app/test/DMServicesTest.cpp
sdm/app/test/GetTimeTest.cpp
sdm/app/test/TaskPostTest.cpp
sdm/app/test/MessageLogAdd.cpp
sdm/app/test/StressxTEDtest.cpp
sdm/app/test/FaultMsgtest.cpp
sdm/app/test/badxTEDS1.cpp
sdm/app/test/BreadBoardTest.cpp
sdm/app/test/MessageLogRemove.cpp
sdm/app/test/SearchTest.cpp
sdm/app/test/VarReq.cpp
sdm/app/test/magtest.cpp
sdm/app/test/messagecountconsumer.cpp
sdm/app/test/VarInfoParserTest.cpp

sdm/app/test/ServicePID.cpp
sdm/app/test/ClassTests/ProviderSubscriptionListTest.cpp
sdm/app/test/ClassTests/SubManTest.cpp
sdm/app/test/ClassTests/MessageManipulatorTest.cpp
sdm/app/test/ClassTests/SDMVarTest.cpp
sdm/app/test/ClassTests/Makefile.uclinux
sdm/app/test/ClassTests/Makefile
sdm/app/test/ClassTests/MessageTests.cpp
sdm/app/test/ClassTests/SDMMessage_IDTest.cpp
sdm/app/test/ClassTests/MessageManipulatorTest.xml
sdm/app/test/ClassTests/TimeTest.cpp
sdm/app/test/PMTests/AppFailSilentTestConsumer.cpp
sdm/app/test/PMTests/Makefile
sdm/app/test/PMTests/AppFailSilentTest.cpp
sdm/app/test/PMTests/AppFailSilentTestProducer.cpp
sdm/app/test/DMTests/SegmentedxTEDSTest.cpp
sdm/app/test/DMTests/SolarArray.xml
sdm/app/test/DMTests/GenericVarReqTest.cpp
sdm/app/test/DMTests/LargexTEDSTest.cpp
sdm/app/test/DMTests/xTEDSPoster.cpp
sdm/app/test/DMTests/xTEDSPoster.h
sdm/app/test/DMTests/Battery.xml
sdm/app/test/DMTests/xTEDSQualifierTests.cpp
sdm/app/test/DMTests/Makefile
sdm/app/test/DMTests/xTEDSReqRegTest.xml
sdm/app/test/DMTests/GenericReqRegTest.cpp
sdm/app/test/DMTests/xTEDSVariableTests.xml
sdm/app/test/DMTests/xTEDSQualifierTests.xml
sdm/app/test/DMTests/xTEDSVariableTests.cpp
sdm/app/test/DMTests/LargexTEDSSmallest.xml
sdm/app/test/DMTests/LargexTEDSSmaller.xml
sdm/app/test/DMTests/LargexTEDS.xml
sdm/app/test/DMTests/GenericSearchTest.cpp
sdm/app/test/DMTests/SubscriptionTests/Makefile
sdm/app/test/DMTests/SubscriptionTests/SubCancelProducer.cpp
sdm/app/test/DMTests/SubscriptionTests/SubCancelTest.cpp
sdm/app/test/DMTests/xTEDSRegTests/PowerController.h
sdm/app/test/DMTests/xTEDSRegTests/ADCSController.xml
sdm/app/test/DMTests/xTEDSRegTests/xTEDSRegTest.cpp
sdm/app/test/DMTests/xTEDSRegTests/CSSAssy.h
sdm/app/test/DMTests/xTEDSRegTests/AnglAccelToTorque.h

sdm/app/test/DMTTests/xTEDSRegTests/SolarArray.xml
sdm/app/test/DMTTests/xTEDSRegTests/Battery.h
sdm/app/test/DMTTests/xTEDSRegTests/ActivityManager.h
sdm/app/test/DMTTests/xTEDSRegTests/StarCamera.xml
sdm/app/test/DMTTests/xTEDSRegTests/SolarArray.h
sdm/app/test/DMTTests/xTEDSRegTests/IRU.xml
sdm/app/test/DMTTests/xTEDSRegTests/Targeting.xml
sdm/app/test/DMTTests/xTEDSRegTests/AnglAccelToTorque.xml
sdm/app/test/DMTTests/xTEDSRegTests/GPS.h
sdm/app/test/DMTTests/xTEDSRegTests/ActivityAgent.xml
sdm/app/test/DMTTests/xTEDSRegTests/Targeting.h
sdm/app/test/DMTTests/xTEDSRegTests/IDS.h
sdm/app/test/DMTTests/xTEDSRegTests/TelemetryHandler.h
sdm/app/test/DMTTests/xTEDSRegTests/DigitalSS.h
sdm/app/test/DMTTests/xTEDSRegTests/ChargeBatteries.xml
sdm/app/test/DMTTests/xTEDSRegTests/AdcoleDigitalSS.h
sdm/app/test/DMTTests/xTEDSRegTests/Battery.xml
sdm/app/test/DMTTests/xTEDSRegTests/GPS_Full.h
sdm/app/test/DMTTests/xTEDSRegTests/MagTorqueRod.xml
sdm/app/test/DMTTests/xTEDSRegTests/ThreeAxisMagnetometer.xml
sdm/app/test/DMTTests/xTEDSRegTests/TelemetryHandler.xml
sdm/app/test/DMTTests/xTEDSRegTests/Makefile
sdm/app/test/DMTTests/xTEDSRegTests/BIT.h
sdm/app/test/DMTTests/xTEDSRegTests/RWheelSingle.xml
sdm/app/test/DMTTests/xTEDSRegTests/GenxTEDS.sh
sdm/app/test/DMTTests/xTEDSRegTests/OOCE.xml
sdm/app/test/DMTTests/xTEDSRegTests/VehicleService.xml
sdm/app/test/DMTTests/xTEDSRegTests/Thruster.h
sdm/app/test/DMTTests/xTEDSRegTests/DownlinkController.h
sdm/app/test/DMTTests/xTEDSRegTests/OOCE.h
sdm/app/test/DMTTests/xTEDSRegTests/iMESA.xml
sdm/app/test/DMTTests/xTEDSRegTests/ThreeAxisMagnetometer.h
sdm/app/test/DMTTests/xTEDSRegTests/iMESA.h
sdm/app/test/DMTTests/xTEDSRegTests/RWheelAssy.xml
sdm/app/test/DMTTests/xTEDSRegTests/GPS_Full.xml
sdm/app/test/DMTTests/xTEDSRegTests/StarCamera.h
sdm/app/test/DMTTests/xTEDSRegTests/ActivityAgent.h
sdm/app/test/DMTTests/xTEDSRegTests/MagTorqueRod.h
sdm/app/test/DMTTests/xTEDSRegTests/VehicleService.h
sdm/app/test/DMTTests/xTEDSRegTests/RoboHub_lean.xml
sdm/app/test/DMTTests/xTEDSRegTests/xteds2str

sdm/app/test/DMTests/xTEDSRegTests/xteds2str.c
sdm/app/test/DMTests/xTEDSRegTests/RoboHub.h
sdm/app/test/DMTests/xTEDSRegTests/ADCSController.h
sdm/app/test/DMTests/xTEDSRegTests/ActivityManager.xml
sdm/app/test/DMTests/xTEDSRegTests/PowerController.xml
sdm/app/test/DMTests/xTEDSRegTests/AdcoleDigitalSS.xml
sdm/app/test/DMTests/xTEDSRegTests/RWheelSingle.h
sdm/app/test/DMTests/xTEDSRegTests/RoboHub.xml
sdm/app/test/DMTests/xTEDSRegTests/Thruster.xml
sdm/app/test/DMTests/xTEDSRegTests/RoboHub_lean.h
sdm/app/test/DMTests/xTEDSRegTests/DownlinkController.xml
sdm/app/test/DMTests/xTEDSRegTests/IRU.h
sdm/app/test/DMTests/xTEDSRegTests/RWheelAssy.h
sdm/app/test/DMTests/xTEDSRegTests/DigitalSS.xml
sdm/app/test/DMTests/xTEDSRegTests/BIT.xml
sdm/app/test/DMTests/xTEDSRegTests/GPS.xml
sdm/app/test/DMTests/xTEDSRegTests/ChargeBatteries.h
sdm/app/test/DMTests/xTEDSRegTests/CSSAssy.xml
sdm/app/test/DMTests/xTEDSRegTests/IDS.xml
sdm/common/ErrorUtils.cpp
sdm/common/sdmLib.h
sdm/common/version.h
sdm/common/UDPcom.h
sdm/common/marshall.h
sdm/common/UDPcom.cpp
sdm/common/sdmLib.cpp
sdm/common/marshall.c
sdm/common/TCPcom.h
sdm/common/SDMComHandle.cpp
sdm/common/SDMCancelQueue.cpp
sdm/common/SDMRegQueue.h
sdm/common/SDMRegQueue.cpp
sdm/common/MemoryUtils.h
sdm/common/Makefile
sdm/common/.wrmakefile
sdm/common/MemoryUtils.c
sdm/common/SDMError.h
sdm/common/asensor.h
sdm/common/.wrproject
sdm/common/SDMCancelQueue.h
sdm/common/Debug.h

sdm/common/TCPcom.cpp
sdm/common/SDMComHandle.h
sdm/common/message_defs.h
sdm/common/ErrorUtils.h
sdm/common/asim.h
sdm/common/.project
sdm/common/message/SDMTaskError.cpp
sdm/common/message/SDMMessage_ID.h
sdm/common/message/SDMSearch.cpp
sdm/common/message/SDMElection.cpp
sdm/common/message/SDMID.h
sdm/common/message/SDMDeletesub.h
sdm/common/message/SDMConsume.cpp
sdm/common/message/SDMCode.h
sdm/common/message/SDMService.cpp
sdm/common/message/SDMAck.cpp
sdm/common/message/SDMxTEDSInfo.h
sdm/common/message/SDMPostTask.cpp
sdm/common/message/SDMTaskFinished.cpp
sdm/common/message/SDMData.cpp
sdm/common/message/SDMReady.cpp
sdm/common/message/SDMCode.cpp
sdm/common/message/SDMReqReg.h
sdm/common/message/SDMHeartbeat.cpp
sdm/common/message/SDMError.cpp
sdm/common/message/SDMCommand.h
sdm/common/message/SDMDMLLeader.cpp
sdm/common/message/SDMTaskFinished.h
sdm/common/message/SDMVarReq.cpp
sdm/common/message/SDMCancel.cpp
sdm/common/message/SDMRegInfo.cpp
sdm/common/message/SDMReqReg.cpp
sdm/common/message/SDMReqCode.cpp
sdm/common/message/Makefile
sdm/common/message/SDMSearch.h
sdm/common/message/SDMRegPM.h
sdm/common/message/SDMTask.cpp
sdm/common/message/SDMCancelxTEDS.cpp
sdm/common/message/SDMRegPM.cpp
sdm/common/message/SDMTask.h
sdm/common/message/SDMData.h

sdm/common/message/SDMCancelxTEDS.h
sdm/common/message/SDMSubreqst.cpp
sdm/common/message/SDMAck.h
sdm/common/message/SDMTat.h
sdm/common/message/SDMSearchReply.cpp
sdm/common/message/SDMDMLLeader.h
sdm/common/message/SDMError.h
sdm/common/message/SDMVarInfo.h
sdm/common/message/SDMCancel.h
sdm/common/message/SDMElection.h
sdm/common/message/SDMxTEDS.cpp
sdm/common/message/SDMVarReq.h
sdm/common/message/SDMmessage.cpp
sdm/common/message/SDMSubreqst.h
sdm/common/message/SDMTaskError.h
sdm/common/message/SDMSearchReply.h
sdm/common/message/SDMRegister.cpp
sdm/common/message/SDMRegInfo.h
sdm/common/message/SDMMessage_ID.cpp
sdm/common/message/SDMSerreqst.h
sdm/common/message/SDMHello.cpp
sdm/common/message/SDMVarInfo.cpp
sdm/common/message/SDMCommand.cpp
sdm/common/message/SDMHeartbeat.h
sdm/common/message/SDMReqCode.h
sdm/common/message/SDMPostTask.h
sdm/common/message/SDMID.cpp
sdm/common/message/SDMService.h
sdm/common/message/SDMTat.cpp
sdm/common/message/SDMReady.h
sdm/common/message/SDMSerreqst.cpp
sdm/common/message/SDMReqxTEDS.h
sdm/common/message/SDMRegister.h
sdm/common/message/SDMmessage.h
sdm/common/message/SDMComponent_ID.h
sdm/common/message/SDMReqxTEDS.cpp
sdm/common/message/SDMComponent_ID.cpp
sdm/common/message/SDMDeletesub.cpp
sdm/common/message/SDMxTEDS.h
sdm/common/message/SDMKill.h
sdm/common/message/SDMConsume.h

sdm/common/message/SDMHello.h
sdm/common/message/SDMxTEDSInfo.cpp
sdm/common/message/SDMKill.cpp
sdm/common/xTEDS/xTEDSOrientationList.cpp
sdm/common/xTEDS/xTEDSMessage.cpp
sdm/common/xTEDS/xTEDSCurve.h
sdm/common/xTEDS/xTEDSVariableList.h
sdm/common/xTEDS/xTEDSParser.h
sdm/common/xTEDS/xTEDSQualifier.cpp
sdm/common/xTEDS/xTEDSSubscription.h
sdm/common/xTEDS/xTEDSLocation.cpp
sdm/common/xTEDS/xTEDSQualifierList.h
sdm/common/xTEDS/xTEDSDrange.cpp
sdm/common/xTEDS/xTEDSCoef.h
sdm/common/xTEDS/lex.xTEDS.c
sdm/common/xTEDS/xTEDSVariable.h
sdm/common/xTEDS/xTEDSFaultMsg.h
sdm/common/xTEDS/MessageDef.h
sdm/common/xTEDS/xTEDSWrapper.cpp
sdm/common/xTEDS/xTEDSDataMsg.h
sdm/common/xTEDS/xTEDS.tab.h
sdm/common/xTEDS/xTEDSLocation.h
sdm/common/xTEDS/xTEDSCurve.cpp
sdm/common/xTEDS/xTEDSNotification.h
sdm/common/xTEDS/xTEDSSubscriptionList.h
sdm/common/xTEDS/xTEDSParser.c
sdm/common/xTEDS/xTEDSOptionList.h
sdm/common/xTEDS/xTEDSQualifier.h
sdm/common/xTEDS/xTEDSOrientationItem.h
sdm/common/xTEDS/xTEDSRequest.cpp
sdm/common/xTEDS/xTEDS.h
sdm/common/xTEDS/xTEDSCommandMsg.cpp
sdm/common/xTEDS/xTEDSNotification.cpp
sdm/common/xTEDS/xTEDSItem.h
sdm/common/xTEDS/xTEDSDataMsg.cpp
sdm/common/xTEDS/xTEDSFaultMsg.cpp
sdm/common/xTEDS/xTEDSDrange.h
sdm/common/xTEDS/Makefile
sdm/common/xTEDS/xTEDSMessage.h
sdm/common/xTEDS/xTEDSOption.h
sdm/common/xTEDS/xTEDS.cpp

sdm/common/xTEDS/xTEDSWrapperList.cpp
sdm/common/xTEDS/xTEDSOrientationList.h
sdm/common/xTEDS/xTEDSOption.cpp
sdm/common/xTEDS/xTEDSOrientationItem.cpp
sdm/common/xTEDS/xTEDSCommandMsg.h
sdm/common/xTEDS/xTEDSVerification.cpp
sdm/common/xTEDS/xTEDSQualifierList.cpp
sdm/common/xTEDS/xTEDSVariableList.cpp
sdm/common/xTEDS/xTEDS.l
sdm/common/xTEDS/xTEDSItemTree.cpp
sdm/common/xTEDS/xTEDSCommand.h
sdm/common/xTEDS/xTEDS.tab.c
sdm/common/xTEDS/xTEDSRequest.h
sdm/common/xTEDS/xTEDSWrapper.h
sdm/common/xTEDS/xTEDSVerification.h
sdm/common/xTEDS/SDMDataTypes.h
sdm/common/xTEDS/VariableDef.cpp
sdm/common/xTEDS/VariableDef.h
sdm/common/xTEDS/xTEDSCoeffList.h
sdm/common/xTEDS/xTEDSCoeffList.cpp
sdm/common/xTEDS/xTEDSItemTree.h
sdm/common/xTEDS/xTEDS.y
sdm/common/xTEDS/xTEDSVariable.cpp
sdm/common/xTEDS/xTEDSOptionList.cpp
sdm/common/xTEDS/xTEDSCoeff.cpp
sdm/common/xTEDS/MessageDef.cpp
sdm/common/xTEDS/xTEDSCommand.cpp
sdm/common/xTEDS/xTEDSItem.cpp
sdm/common/xTEDS/SDMDataRates.h
sdm/common/xTEDS/xTEDSWrapperList.h
sdm/common/Regex/RegexMatch.h
sdm/common/Regex/RegularExpression.cpp
sdm/common/Regex/Regex.cpp
sdm/common/Regex/Makefile
sdm/common/Regex/RegexResult.cpp
sdm/common/Regex/RegexCapture.cpp
sdm/common/Regex/Regex.h
sdm/common/Regex/RegexResult.h
sdm/common/Regex/RegexMatch.cpp
sdm/common/Regex/RegexCapture.h
sdm/common/Regex/RegularExpression.h

sdm/common/Time/TimeKeepingWin32.cpp
sdm/common/Time/TimeKeepingLinux.cpp
sdm/common/Time/SDMTime.cpp
sdm/common/Time/LookupTable.h
sdm/common/Time/TimerList.cpp
sdm/common/Time/SDMTimeLinux.cpp
sdm/common/Time/TimerList.h
sdm/common/Time/Makefile
sdm/common/Time/SDMTimeWin32.h
sdm/common/Time/TimeKeeping.h
sdm/common/Time/SDMTimeWin32.cpp
sdm/common/Time/SecTime.h
sdm/common/Time/SecTime.cpp
sdm/common/Time/SDMTime.h
sdm/common/Time/SDMTimeLinux.h
sdm/common/Time/LookupTable.cpp
sdm/common/MessageManipulator/msgdef.y
sdm/common/MessageManipulator/message.h
sdm/common/MessageManipulator/message.c
sdm/common/MessageManipulator/msgdef.tab.c
sdm/common/MessageManipulator/lex.MessageManipulator.c
sdm/common/MessageManipulator/Makefile
sdm/common/MessageManipulator/msgdef.l
sdm/common/MessageManipulator/MessageManipulator.h
sdm/common/MessageManipulator/msgdef.tab.h
sdm/common/MessageManipulator/MessageManipulator.cpp
sdm/common/MessageLogger/SDMMessageLogger.h
sdm/common/MessageLogger/Makefile
sdm/common/MessageLogger/MessageLogger.cpp
sdm/common/MessageLogger/MessageLogger.h
sdm/common/MessageLogger/SDMMessageLogger.cpp
sdm/common/checksum/crcTable.c
sdm/common/checksum/checksum.c
sdm/common/checksum/crcmodel.h
sdm/common/checksum/crcmodel.c
sdm/common/checksum/Makefile
sdm/common/checksum/checksum.h
sdm/common/SubscriptionManager/SubscriptionManager.h
sdm/common/SubscriptionManager/SubscriptionManager.cpp
sdm/common/SubscriptionManager/Makefile
sdm/common/MessageManager/Makefile

sdm/common/MessageManager/MessageManager.cpp
sdm/common/MessageManager/MessageManager.h
sdm/common/task/SDMTaskResources.cpp
sdm/common/task/Makefile
sdm/common/task/tasklist.h
sdm/common/task/SDMTaskResources.h
sdm/common/task/tasklist.cpp
sdm/common/task/task.cpp
sdm/common/task/task.h
sdm/common/task/taskdefs.h
sdm/common/VarInfoParser/Variable.h
sdm/common/VarInfoParser/Variable.c
sdm/common/VarInfoParser/lex.VarInfoParser.c
sdm/common/VarInfoParser/VarInfoParser.y
sdm/common/VarInfoParser/Makefile
sdm/common/VarInfoParser/VarInfoParser.tab.c
sdm/common/VarInfoParser/VarInfoParser.cpp
sdm/common/VarInfoParser/VarInfoParser.l
sdm/common/VarInfoParser/VarInfoParser.h
sdm/common/VarInfoParser/VarInfoParser.tab.h
sdm/common/Exception/SDMException.h
sdm/common/Exception/SDMBadIndexException.cpp
sdm/common/Exception/Makefile
sdm/common/Exception/SDMException.cpp
sdm/common/Exception/SDMRegexException.h
sdm/common/Exception/SDMBadIndexException.h
sdm/common/Exception/SDMRegexException.cpp
sdm/common/asim/asim_win32.h
sdm/common/asim/ASIM.cpp
sdm/common/asim/ASIM.h
sdm/common/asim/Makefile
sdm/common/asim/asim_win32.cpp
sdm/common/asim/asim_commands.h
sdm/common/semaphore/semaphore.h
sdm/common/semaphore/semaphore.cpp
sdm/common/semaphore/Makefile
sdm/Win32/Win32.sln
sdm/Win32/Win32.vcproj
sdm/Win32/converter/converter.vcproj
sdm/Win32/pm_ids_code_transfer/pm_ids_code_transfer.vcproj
sdm/Win32/sm_monitor/sm_monitor.vcproj

sdm/Win32/unix/semaphore.h
sdm/Win32/unix/poll.cpp
sdm/Win32/unix/endian.h
sdm/Win32/unix/pthreadVSE2.lib
sdm/Win32/unix/sched.h
sdm/Win32/unix/socket.cpp
sdm/Win32/unix/pthreadVC2.lib
sdm/Win32/unix/pthread.h
sdm/Win32/unix/stdint.h
sdm/Win32/unix/getopt.cpp
sdm/Win32/unix/unistd.h
sdm/Win32/unix/unix.cpp
sdm/Win32/unix/getopt.h
sdm/Win32/unix/time.cpp
sdm/Win32/unix/byteswap.h
sdm/Win32/unix/sem.cpp
sdm/Win32/unix/unix/fcntl.h
sdm/Win32/unix/arpa/inet.h
sdm/Win32/unix/sys/ipc.h
sdm/Win32/unix/sys/socket.h
sdm/Win32/unix/sys/time.h
sdm/Win32/unix/sys/poll.h
sdm/Win32/unix/sys/ioctl.h
sdm/Win32/unix/sys/sem.h
sdm/Win32/unix/sys/wait.h
sdm/Win32/unix/netinet/in.h
sdm/Win32/pm_ids/pm_ids.vcproj
sdm/Win32/PM_Fork_Win32/PM_Fork_Win32.vcproj
sdm/Win32/Regex/regex.lib
sdm/Win32/Regex/regex2.dll
sdm/Win32/Regex/regex.h
sdm/Win32/pm/pm.vcproj
sdm/Win32/SearchTest/SearchTest.vcproj
sdm/Win32/sm/sm.vcproj
sdm/Win32/bin/sleep.exe
sdm/Win32/bin/regex2.dll
sdm/Win32/bin/pthreadVC2.dll
sdm/Win32/bin/pthreadVC.dll
sdm/Win32/tm/tm.vcproj
sdm/Win32/dm/dm.vcproj
sdm/Win32/tm_monitor/tm_monitor.vcproj

sdm/Win32/sdmLib/sdmLib.cpp
sdm/Win32/sdmLib/sdmLib.vcproj
sdm/Win32/xTEDSUpdateTest/xTEDSUpdateTest.vcproj
sdm/Win32/producer/producer.vcproj
sdm/Win32/consumer/consumer.vcproj

List of files not included in source listing

sdm/pm/.wrmakefile
sdm/pm/.wrproject
sdm/pm/.project
sdm/tm/.wrmakefile
sdm/tm/.wrproject
sdm/tm/.project
sdm/dm/.wrmakefile
sdm/dm/.wrproject
sdm/dm/.project
sdm/VxWorks/bin/regex/libRegex.a
sdm/VxWorks/libRegex/.wrmakefile
sdm/VxWorks/libRegex/.wrproject
sdm/VxWorks/libRegex/.project
sdm/VxWorks/apps/converter/.wrmakefile
sdm/VxWorks/apps/converter/.wrproject
sdm/VxWorks/apps/converter/.project
sdm/VxWorks/apps/producer/.wrmakefile
sdm/VxWorks/apps/producer/.wrproject
sdm/VxWorks/apps/producer/.project
sdm/VxWorks/apps/consumer/.wrmakefile
sdm/VxWorks/apps/consumer/.wrproject
sdm/VxWorks/apps/consumer/.project
sdm/app/test/DMTTests/xTEDSRegTests/xteds2str
sdm/common/.wrmakefile
sdm/common/.wrproject
sdm/common/.project
sdm/Win32/Win32.sln
sdm/Win32/Win32.vcproj
sdm/Win32/converter/converter.vcproj
sdm/Win32/pm_ids_code_transfer/pm_ids_code_transfer.vcproj
sdm/Win32/sm_monitor/sm_monitor.vcproj
sdm/Win32/unix/pthreadVSE2.lib
sdm/Win32/unix/pthreadVC2.lib
sdm/Win32/pm_ids/pm_ids.vcproj

sdm/Win32/PM_Fork_Win32/PM_Fork_Win32.vcproj
sdm/Win32/Regex/regex.lib
sdm/Win32/Regex/regex2.dll
sdm/Win32/pm/pm.vcproj
sdm/Win32/SearchTest/SearchTest.vcproj
sdm/Win32/sm/sm.vcproj
sdm/Win32/bin/sleep.exe
sdm/Win32/bin/regex2.dll
sdm/Win32/bin/pthreadVC2.dll
sdm/Win32/bin/pthreadVC.dll
sdm/Win32/tm/tm.vcproj
sdm/Win32/dm/dm.vcproj
sdm/Win32/tm_monitor/tm_monitor.vcproj
sdm/Win32/sdmLib/sdmLib.vcproj
sdm/Win32/xTEDSUpdateTest/xTEDSUpdateTest.vcproj
sdm/Win32/producer/producer.vcproj
sdm/Win32/consumer/consumer.vcproj

Source listing

File: sdm/Makefile.defs

```
1: # -*- Makefile -*-
2: # Common definitions shared by all Makefiles in the SDM tree.
3: # These definitions are collected here for easy modification.
4:
5: # The C and C++ compilers to use.
6: # If a cross compiler is being used, set the environment variable CROSS_COMPILE (using export) to
the path
7: #CROSS_COMPILE=/opt/gumstix/build_arm_nofpu/staging_dir/bin/arm-linux-uclibcgnueabi-
8: CC=$(CROSS_COMPILE)gcc
9: CXX=$(CROSS_COMPILE)g++
10: AR=$(CROSS_COMPILE)ar
11:
12: # Architecture-specific compiler flags
13: #ARCHFLAGS=-m32 -march=i686
14: ARCHFLAGS=
15: #ARCHFLAGS=-DWSSP_BUILD
16: #ARCHFLAGS=-Os -march=armv5te -mtune=xscale -Wa,-mcpu=xscale
17:
18: # Warning flags for both the C and C++ compilers
19: WARNFLAGS=-Wall -W -Wno-unused-parameter
```

```

20:
21: # Warning flags for the C++ compiler only
22: CXXWARNFLAGS=-Weffc++
23:
24: # Optimization flags
25: OPTFLAGS=
26:
27: # Debug flags
28: DEBUGFLAGS=-g
29:
30: # The lexer to invoke and flags to pass to it
31: LEX=flex
32: LEXFLAGS=-B
33:
34: # The parser generator to invoke and flags to pass to it
35: YACC=bison
36: YACCFLAGS=-vkdtd
37:
38: # Build to log all SDM message traffic, uncomment the below variable to log SDM messages
39: #MESSAGELOGGINGFLAGS=-DBUILD_WITH_MESSAGE_LOGGING
40:
41: # Uncomment and add the below variable to the build definitions to remove debug statements
42: #DEBUGREMOVALFLAGS=-DREMOVE_DEBUG_OUTPUT
43:
44: # Uncomment to allow backup Data Managers and backup Task Managers for increased fault
tolerance
45: #DMBACKUPFLAGS=-DPNP_BACKUP
46:
47: # Uncomment to allow backup Data Managers for increased fault tolerance using a fake NM
48: #DMFAKEBACKUPFLAGS=-DPNP_FAKE
49:
50: # Uncomment to build the Data Manager for xTEDS merging
51: #DMMERGEFLAGS=-DBUILD_FOR_XTEDS_MERGING
52:
53: # Uncomment to use spacewire
54: #DSPACEWIREFLAGS=-DBUILD_FOR_SPACEWIRE
55:
56: # Do not change these definitions
57: CFLAGS=$(ARCHFLAGS) $(WARNFLAGS) $(OPTFLAGS) $(DEBUGFLAGS)
58: CXXFLAGS=$(CFLAGS) $(CXXWARNFLAGS)
59:

```

File: sdm/Makefile.defs.uclinux

```
1: # -*- Makefile -*-
2:          ##                      Section                      for                      uclinux
#####
3: FLTFLAGS += -s 4194304
4:
5: ## The lexer to invoke and flags to pass to it
6: LEX=flex
7: LEXFLAGS=-B
8:
9: ## The parser generator to invoke and flags to pass to it
10: YACC=bison
11: YACCFLAGS=-vkdtd
12:
13: CFLAGS +=-DSEND_IMA
14: CXXFLAGS+=-DSEND_IMA
15:
16: ## Include the boost libraries, these 2 lines can be commented out to remove the use of the boost
libraries
17: ##BOOST=-DBOOST
18: ##BOOSTFLAGS=-lboost_regex
19:
20: ## Build to log all SDM message traffic, uncomment the below variable to log SDM messages
21: ##MESSAGELOGGING=-DBUILD_WITH_MESSAGE_LOGGING
22:
23: # Uncomment and add the below variable to the build definitions to remove debug statements
24: #DEBUGREMOVALFLAGS=-DREMOVE_DEBUG_OUTPUT
25:
26: # Uncomment to allow backup Data Managers for increased fault tolerance
27: #DMBACKUPFLAGS=-DPNP_BACKUP
28:
```

File: sdm/Makefile.common

```
1: ifndef MAKEFILE_DEFS
2: MAKEFILE_DEFS=Makefile.defs
3: export MAKEFILE_DEFS
4: endif
```

File: sdm/Makefile.uclinux

```
1: ifndef PETALINUX
2: $(error You must source the petalinux/settings.sh script before working with PetaLinux)
3: endif
4:
5: # Point to default PetaLinux root directory
6: ifndef ROOTDIR
7: ROOTDIR=$(PETALINUX)/software/petalinux-dist
8: endif
9:
10: PATH:=$(PATH):$(ROOTDIR)/tools
11:
12: UCLINUX_BUILD_USER = 1
13: -include $(ROOTDIR)/.config
14: -include $(ROOTDIR)/$(CONFIG_LINUXDIR)/.config
15: LIBCDIR = $(CONFIG_LIBCDIR)
16: -include $(ROOTDIR)/config.arch
17: ROMFSDIR=$(ROOTDIR)/romfs
18: ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh
19:
20: MAKEFILE_DEFS=Makefile.defs.uclinux
21: export MAKEFILE_DEFS
22: APPFOLDERS_BUILD=dm tm pm app/examples
23: APPFOLDERS_ROMFS=dm tm pm
24:
25: all: libSDM subdirs
26:
27: subdirs:
28: for dir in $(APPFOLDERS_BUILD); do \
29:     make -C $$dir -f Makefile.uclinux; \
30: done
31:
32: libSDM:
33: make -C common uclinux
34:
35: examples: libSDM
36: make -C app/examples all
37:
38: clean:
39: for dir in $(APPFOLDERS_BUILD); do \
```



```

40:     make -C $$dir -f Makefile.uclinux clean; \
41: done
42: make -C common clean
43:
44: romfs: libSDM subdirs
45: for dir in $(APPFOLDERS_ROMFS); do \
46:     make -C $$dir -f Makefile.uclinux romfs; \
47: done
48:
49: #   $(ROMFSINST) $(APP) /bin/$(APP)
50:
51: %.o: %.c
52: $(CC) -c $(CFLAGS) -o $$@ $$<
53:
54:
55: # Targets for the required .config files - if they don't exist, the tree isn't
56: # configured. Tell the user this, how to fix it, and exit.
57: ${ROOTDIR}/config.arch ${ROOTDIR}/.config:
58: @echo "Error: You must configure the PetaLinux tree before compiling your application"
59: @echo ""
60: @echo "Change directory to ../../petalinux-dist and 'make menuconfig' or 'make xconfig'"
61: @echo ""
62: @echo "Once the tree is configured, return to this directory, and re-run make."
63: @echo ""
64: @exit -1
65:

```

File: sdm/Makefile

```
1: # Top-level Makefile
2:
3: MAKEFILE_DEFS=Makefile.defs
4: export MAKEFILE_DEFS
5:
6: include $(MAKEFILE_DEFS)
7:
8: .PHONY:    all clean distclean package
9:
10: SUBDIRS=common app dm pm sm tm Spa1Manager
11:
12: all:
13: for dir in $(SUBDIRS); do \
14:     make -S -C $$dir || exit 1; \
15: done
16:
17: clean:
18: for dir in $(SUBDIRS); do \
19:     make -C $$dir clean; \
20: done
21:
22: distclean:
23: for dir in $(SUBDIRS); do \
24:     make -C $$dir distclean; \
25: done
26:
27: package:    SDMSources.tar.gz SDMSourcesLinux.tar.gz SDMBinaries.tar.gz SDMExamples.tar.gz
28:
29: ASIMTester.tar.gz:
30: tar -cz -X asim_tester_exclude * > $@
31:
32: SDMExamples.tar.gz:
33: tar -cz -X example_package_exclude * > $@
34:
35: SDMSources.tar.gz:
36: tar -cz -X source_package_exclude * > $@
37:
38: SDMSourcesLinux.tar.gz:
39: tar -cz -X source_package_linux_exclude * > $@
```

```
40:
41: SDMBinaries.tar.gz:all
42: mkdir bin
43: mkdir lib
44: cp ./dm/dm ./bin/.
45: cp ./tm/tm ./bin/.
46: cp ./tm/tm_process ./bin/.
47: cp ./pm/pm ./bin/.
48: cp ./sm/sm ./bin/.
49: cp ./sm/sm_process ./bin/.
50: cp ./common/libSDM.so.1.0 ./lib/libSDM.so
51: tar -cz bin lib > $@
52: rm ./bin/*
53: rm ./lib/*
54: rmdir bin
55: rmdir lib
56:
```

Listing from directory: sdm/app

File: sdm/app/Makefile

```
1: # App Makefile
2:
3: include ../Makefile.defs
4:
5: .PHONY:    all clean distclean
6:
7: SUBDIRS=examples
8: SUBDIRS_CLEAN= $(SUBDIRS) test
9:
10: all:
11: for dir in $(SUBDIRS); do \
12:     make -C $$dir; \
13: done
14:
15: clean:
16: for dir in $(SUBDIRS_CLEAN); do \
17:     make -C $$dir clean; \
18: done
19:
20: distclean:
21: for dir in $(SUBDIRS_CLEAN); do \
22:     make -C $$dir distclean; \
23: done
```

File: sdm/app/examples/Makefile.uclinux

```
1: ifndef PETALINUX
2: $(error You must source the petalinux/settings.sh script before working with PetaLinux)
3: endif
4:
5: # Point to default PetaLinux root directory
6: ifndef ROOTDIR
7: ROOTDIR=$(PETALINUX)/software/petalinux-dist
8: endif
9:
10: PATH:=$(PATH):$(ROOTDIR)/tools
11:
12: UCLINUX_BUILD_USER = 1
13: -include $(ROOTDIR)/.config
14: -include $(ROOTDIR)/$(CONFIG_LINUXDIR)/.config
15: LIBCDIR = $(CONFIG_LIBCDIR)
16: -include $(ROOTDIR)/config.arch
17: ROMFSDIR=$(ROOTDIR)/romfs
18: ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh
19:
20: FLTFLAGS+=-s 262144
21: export FLTFLAGS
22:
23: LDLIBS += -lSDM -lpthread -lstdc++
24: LDFLAGS += -L../common/
25:
26: EXAMPLES = converter consumer producer
27: all: $(EXAMPLES)
28:
29: $(APP): $(APP_OBJS)
30: $(CXX) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)
31:
32: consumer: consumer.o
33: $(CXX) $(LDFLAGS) -o $@ $^ $(LDLIBS)
34:
35: producer: producer.o
36: $(CXX) $(LDFLAGS) -o $@ $^ $(LDLIBS)
37:
38: converter: converter.o
39: $(CXX) $(LDFLAGS) -o $@ $^ $(LDLIBS)
```

```

40:
41: clean:
42: -rm -f $(EXAMPLES) *.elf *.gdb *.o
43:
44: romfs:
45: for app in $(EXAMPLES); do \
46:     $(ROMFSINST) $$app /bin/$$app; \
47: done
48:
49: %.o: %.cpp
50: $(CXX) -c $(CXXFLAGS) -o $@ $<
51:
52:
53: # Targets for the required .config files - if they don't exist, the tree isn't
54: # configured. Tell the user this, how to fix it, and exit.
55: ${ROOTDIR}/config.arch ${ROOTDIR}/.config:
56: @echo "Error: You must configure the PetaLinux tree before compiling your application"
57: @echo ""
58: @echo "Change directory to ../../petalinux-dist and 'make menuconfig' or 'make xconfig'"
59: @echo ""
60: @echo "Once the tree is configured, return to this directory, and re-run make."
61: @echo ""
62: @exit -1
63:

```

File: sdm/app/examples/Makefile

```
1: include ../../Makefile.defs
2:
3: .PHONY:    clean distclean
4:
5: all: consumer converter producer copy
6:
7: producer: producer.o
8: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
9:
10: converter: converter.o
11: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
12:
13: consumer: consumer.o
14: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
15:
16: %.o:    %.cpp
17: $(CXX) $(CXXFLAGS) -c $<
18:
19: copy:
20: cp producer ../../tm
21: cp consumer ../../tm
22: cp converter ../../tm
23:
24: clean:
25: rm -f *.o *~
26:
27: distclean: clean
28: rm -f consumer converter producer
29: rm -f ../../tm/consumer ../../tm/converter ../../tm/producer
30: rm -f ../../pm/consumer ../../pm/converter ../../pm/producer
```

File: sdm/app/examples/producer.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMSubreqst.h"
3: #include "../common/message/SDMDeletesub.h"
4: #include "../common/message/SDMCancelxTEDS.h"
5: #include "../common/SubscriptionManager/SubscriptionManager.h"
6: #include "../common/MessageManager/MessageManager.h"
7: #include "../common/message/SDMHeartbeat.h"
8: #include <string.h>
9: #include <sys/types.h>
10: #include <sys/stat.h>
11: #include <fcntl.h>
12: #include <unistd.h>
13: #include <stdio.h>
14: #include <stdlib.h>
15: #include <pthread.h>
16:
17: const char* XML_HEADER = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n";
18: const char* XTEDS_HEADER = "<xTEDS version= \"2.0 \" name= \"Producer_xTEDS \">>";
19: const char* APP_SECTION = " \n \t<Application name= \"producer \" kind= \"data \"/>";
20: const char* INTERFACE_SECTION = " \n \t<Interface name= \"Producer_Interface \" id= \"1 \">>";
21: const char* VAR_DATA_1 = " \n \t<Variable name= \"data \" format= \"UINT16 \" ";
22: const char* VAR_DATA_2 = "kind= \"data \"/>";
23: const char* NOTIFICATION = " \n \t<Notification>";
24: const char* MSG_ALL_1 = " \n \t \t<DataMsg name= \"all \" id= \"1 \" ";
25: const char* MSG_ALL_2 = "msgArrival= \"PERIODIC \" msgRate= \"1 \">>";
26: const char* MSG_ALL_3 = " \n \t \t \t<VariableRef name= \"data \"/>";
27: const char* MSG_ALL_4 = " \n \t \t</DataMsg>";
28: const char* NOTIFICATION_END = " \n \t</Notification>";
29: const char* INTERFACE_END = " \n \t</Interface>";
30: const char* XTEDS_END = " \n</xTEDS> \n";
31:
32: void RegisterxTEDS();
33: void CancelxTEDS();
34: void* Publisher(void *);
35: void* Listener(void *);
36:
37: SubscriptionManager subscriptions;
38: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER;
39: long my_port;
```



```

40: const unsigned int THREAD_STACK_SIZE = 128000;
41:
42: int main(int argc, char** argv)
43: {
44:     pthread_t ListenerThread;
45:     pthread_t PublisherThread;
46:
47:     SDMInit(argc, argv);
48:     my_port = getPort();
49:     if(my_port == SDM_PM_NOT_AVAILABLE)
50:     {
51:         printf("No PM is available to get port from! \n");
52:         return 0;
53:     }
54:
55:     pthread_attr_t threadAttr;
56:     pthread_attr_init(&threadAttr);
57:     pthread_attr_setstacksize(&threadAttr, THREAD_STACK_SIZE);
58:
59:     pthread_create(&ListenerThread, &threadAttr, &Listener, NULL);
60:     //usleep(100);
61:     RegisterxTEDS();
62:     pthread_create(&PublisherThread, &threadAttr, &Publisher, NULL);
63:     pthread_join(PublisherThread, NULL);
64:     CancelxTEDS();
65:     pthread_cancel(ListenerThread);
66:     pthread_join(ListenerThread, NULL);
67: }
68:
69: void* Publisher(void * args)
70: {
71:     int published = 0;
72:     short data;
73:     while(published < 10)
74:     {
75:         data = (short)(rand() & 0x00FF);
76:         char bufdata[2];
77:         PUT_SHORT(bufdata, data);
78:         pthread_mutex_lock(&subscription_mutex);
79:         if (subscriptions.Publish(1, 1, bufdata, 2))
80:         {

```

```

81:         published++;
82:     }
83:     pthread_mutex_unlock(&subscription_mutex);
84:     printf("Produced %d \tPublished %d / 10 \n",data,published);
85:     sleep(1);
86: }
87: return NULL;
88: }
89:
90: void* Listener(void * args)
91: {
92:     char buf[BUFSIZE];
93:     SDMSubreqst sub;
94:     SDMDeletesub del;
95:     MessageManager mm;
96:     mm.Async_Init(my_port);
97:     while(1)
98:     {
99:         pthread_testcancel();
100:         if(mm.IsReady())
101:         {
102:             SendHeartbeat();
103: #ifdef WIN32
104:             switch(mm.GetMsg(buf))
105: #else
106:             switch(mm.GetMessage(buf))
107: #endif
108:             {
109:                 case SDM_Subreqst:
110:                     sub.Unmarshal(buf);
111:                     printf("Subscription Rec'd for %d \n",sub.msg_id.getInterfaceMessagePair());
112:                     pthread_mutex_lock(&subscription_mutex);
113:                     subscriptions.AddSubscription(sub);
114:                     pthread_mutex_unlock(&subscription_mutex);
115:                     break;
116:                 case SDM_Deletesub:
117:                     printf("Cancel Rec'd \n");
118:                     del.Unmarshal(buf);
119:                     pthread_mutex_lock(&subscription_mutex);
120:                     subscriptions.RemoveSubscription(del);

```

```

121:         pthread_mutex_unlock(&subscription_mutex);
122:         break;
123:     default:
124:         printf("Invalid Message found! \n");
125:         fflush(NULL);
126:         break;
127:     }
128: }
129: else
130: {
131:     usleep(100000);
132: }
133: }
134: return NULL;
135: }
136:
137: void RegisterxTEDS()
138: {
139:     // create an xTEDS registration message
140:     SDMxTEDS xteds;
141:
142:     // set xTEDS
143:     strcat (xteds.xTEDS,XML_HEADER);
144:     strcat (xteds.xTEDS,XTEDS_HEADER);
145:     strcat (xteds.xTEDS,APP_SECTION);
146:     strcat (xteds.xTEDS,INTERFACE_SECTION);
147:     strcat (xteds.xTEDS,VAR_DATA_1);
148:     strcat (xteds.xTEDS,VAR_DATA_2);
149:     strcat (xteds.xTEDS,NOTIFICATION);
150:     strcat (xteds.xTEDS,MSG_ALL_1);
151:     strcat (xteds.xTEDS,MSG_ALL_2);
152:     strcat (xteds.xTEDS,MSG_ALL_3);
153:     strcat (xteds.xTEDS,MSG_ALL_4);
154:     strcat (xteds.xTEDS,NOTIFICATION_END);
155:     strcat (xteds.xTEDS,INTERFACE_END);
156:     strcat (xteds.xTEDS,XTEDS_END);
157:
158:     // set the id of this application
159:     xteds.source.setSensorID(1);
160:     xteds.source.setPort(my_port);
161:     printf("Registering producer xTEDS on port %ld \n",my_port);

```

```
162:    // register with the SDM
163:    xteds.Send();
164: }
165:
166: void CancelxTEDS()
167: {
168:     SDMCancelxTEDS cancel;
169:     printf("Canceling xTEDS \n");
170:     cancel.source.setSensorID(1);
171:     cancel.source.setPort(my_port);
172:     cancel.Send();
173: }
```

```

1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMmessage.h"
3: #include "../common/message/SDMSerreqst.h"
4: #include "../common/message/SDMData.h"
5: #include "../common/message/SDMCancelxTEDS.h"
6: #include "../common/message/SDMHeartbeat.h"
7: #include <string.h>
8: #include <sys/types.h>
9: #include <sys/stat.h>
10: #include <fcntl.h>
11: #include <unistd.h>
12: #include <stdio.h>
13:
14: long my_port;
15:
16: //xTEDS data
17: const char* XML_HEADER = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?>";
18: const char* XTEDS_SECTION = " \n<xTEDS version= \"2.0 \" name= \"converter_xTEDS \"/>";
19: const char* APP_SECTION = " \n \t<Application name= \"converter \" kind= \"Software \"/>";
20: const char* INTERFACE = " \n \t<Interface name= \"Converter_Interface \" id= \"1 \"/>";
21: const char* VAR_DATA_1 = " \n \t \t<Variable name= \"data \" format= \"FLOAT32 \"";
22: const char* VAR_DATA_2 = " kind= \"Float_Data \"/>";
23: const char* VAR_CONVERTEE_1 = " \n \t \t<Variable name= \"converttee \" format= \"UINT16 \"";
24: const char* VAR_CONVERTEE_2 = " kind= \"Int_Data \"/>";
25: const char* REQUEST = " \n \n \t \t<Request>";
26: const char* CMD_CONVERT_1 = " \n \t \t \t<CommandMsg name= \"convert \" id= \"1 \"/>";
27: const char* CMD_CONVERT_2 = " \n \t \t \t \t<VariableRef name= \"converttee \"/>";
28: const char* CMD_CONVERT_3 = " \n \t \t \t</CommandMsg>";
29: const char* MSG_RESULTS_1 = " \n \t \t \t<DataReplyMsg name= \"results \" id= \"2 \"";
30: const char* MSG_RESULTS_2 = " msgArrival= \"EVENT \"> \n \t \t \t \t";
31: const char* MSG_RESULTS_3 = "<VariableRef name= \"data \"/> \n \t \t \t</DataReplyMsg>";
32: const char* REQUEST_END = " \n \t \t</Request>";
33: const char* INTERFACE_END = " \n \t</Interface>";
34: const char* XTEDS_END = " \n</xTEDS>";
35:
36: int main(int argc,char** argv)
37: {
38:
39: SDMInit(argc,argv);

```

```

40: my_port = getPort();
41: if(my_port == SDM_PM_NOT_AVAILABLE)
42: {
43:     printf("No PM is available to get port from! \n");
44:     return 0;
45: }
46: unsigned short int_data;
47: float float_data;
48: int i;
49:
50: // create an xTEDS registration message
51: SDMxTEDS xteds;
52: // create a message to receive a service message request
53: SDMSerreqst request;
54: // create a message to receive a data message
55: SDMData dat;
56: SDMCancelxTEDS cancel;
57:
58: // set xTEDS
59: strcat (xteds.xTEDS,XML_HEADER);
60: strcat (xteds.xTEDS,XTEDS_SECTION);
61: strcat (xteds.xTEDS,APP_SECTION);
62: strcat (xteds.xTEDS,INTERFACE);
63: strcat (xteds.xTEDS,VAR_DATA_1);
64: strcat (xteds.xTEDS,VAR_DATA_2);
65: strcat (xteds.xTEDS,VAR_CONVERTEE_1);
66: strcat (xteds.xTEDS,VAR_CONVERTEE_2);
67: strcat (xteds.xTEDS,REQUEST);
68: strcat (xteds.xTEDS,CMD_CONVERT_1);
69: strcat (xteds.xTEDS,CMD_CONVERT_2);
70: strcat (xteds.xTEDS,CMD_CONVERT_3);
71: strcat (xteds.xTEDS,MSG_RESULTS_1);
72: strcat (xteds.xTEDS,MSG_RESULTS_2);
73: strcat (xteds.xTEDS,MSG_RESULTS_3);
74: strcat (xteds.xTEDS,REQUEST_END);
75: strcat (xteds.xTEDS,INTERFACE_END);
76: strcat (xteds.xTEDS,XTEDS_END);
77:
78: // set the id of this application
79: xteds.source.setSensorID(1);
80: cancel.source.setSensorID(xteds.source.getSensorID());

```

```

81: cancel.source.setPort(my_port);
82: xteds.source.setPort(my_port);
83: printf("Registering converter xTEDS on port %ld \n",my_port);
84: // register with the SDM
85: xteds.Send();
86:
87: // now wait for and handle 15 service request messages
88: for (i=0;i<15;i++)
89: {
90:     printf("Waiting for service request. \n");
91:     request.Recv(my_port);
92:     printf("Request Received");
93:     SendHeartbeat();
94:
95: // request contains the service request parameters:
96: // request.service = "convert" and 2 raw data bytes
97: // representing a unsigned integer strain.
98:
99: // The following converts the raw parameter bytes
100: // into a float and copies it into variable temp.
101:
102:     // copy integer parameter into variable int_strain
103:     int_data = GET_SHORT(request.data);
104:     // convert to float between 0 and 1
105:     float_data = int_data/(float)0xffff;
106:     printf("Converted %d to %f \n",int_data,float_data);
107:     dat.source = request.source;
108:     // fill in other fields
109:     dat.msg_id = request.reply_id;
110:     // convert float_data to raw data bytes
111:     PUT_FLOAT(dat.msg, float_data);
112:     printf("Sending reply. \n");
113:     // return to requester
114:     dat.Send (request.destination,4);
115: }
116: printf("Canceling xTEDS \n");
117: cancel.Send();
118:
119: }

```

File: sdm/app/examples/consumer.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMHeartbeat.h"
7: #include "../common/MessageManipulator/MessageManipulator.h"
8: #include "../common/MessageManager/MessageManager.h"
9:
10: #include <string.h>
11: #include <unistd.h>
12: #include <stdio.h>
13: #include <sys/types.h>
14: #include <sys/wait.h>
15: #include <signal.h>
16:
17: #define DATA_PROVIDER    1
18: #define SERVICE_PROVIDER 2
19:
20: long my_port;
21: SDMComponent_ID data_provider;
22: SDMComponent_ID service_provider;
23: SDMMessage_ID data_msg(0,0);
24: SDMMessage_ID service_msg(0,0);
25:
26: MessageManipulator data_manipulator;
27: MessageManipulator service_manipulator;
28:
29: void DataHandler(SDMData& dat,long length)
30: {
31:     SDMService request;
32:     short int_value;
33:     float float_value;
34:     static float geo_average = 0;
35:
36:     if((dat.source == data_provider)&&(dat.msg_id == data_msg))
37:     {
38:         //marshal appropriate service message
39:         //copy source componentID into service request
```



```

40:     request.source=service_provider;
41:     request.destination.setPort(my_port);
42:     //copy msg id into service request
43:     request.command_id=service_msg;
44:     //set the length we are sending
45:     request.length = sizeof(short);
46:     //copy integer data into service request
47:     if(service_provider.getSensorID()==0)
48:     {
49:         printf("No converter is available \n");
50:         return;
51:     }
52:     int_value = data_manipulator.getUINT16Value("data",dat,DATAMSG);
53:     service_manipulator.setValue("convertee",request,int_value);
54:     //send request
55:     request.Send();
56: }
57: else if((dat.source == service_provider))
58: {
59:     float_value = service_manipulator.getFloat32Value("data",dat,DATAMSG);
60:     geo_average /=2;
61:     geo_average += float_value;
62:     //extract and display results
63:     printf("Running Average -- %f \n",geo_average);
64: }
65: }
66:
67: void RegInfoHandler(SDMRegInfo& info)
68: {
69:     SDMConsume consume;
70:     SDMReqReg req_reg;
71:
72:     //Set the port we will be receiving on
73:     consume.destination.setPort(my_port);
74:     //copy the sensor id into the consume message
75:     consume.source=info.source;
76:     //copy the msg id into the consume message
77:     consume.msg_id=info.msg_id;
78:
79:     switch(info.id)
80:     {

```

```

81: case DATA_PROVIDER:
82:     if(info.type == 1)
83:     {
84:         data_provider.setSensorID(0);
85:         printf("Data provider failed . . . ");
86:         printf("Searching for new provider \n");
87:         //request info on integer data providers,
88:         //the DM will repost software tasks that
89:         // could satisfiy our requirments
90:
91:         //Set variable name
92:         strcpy(req_reg.item_name,"data");
93:         //Set the quallist can be empty
94:         strcpy(req_reg.quallist,"< format= \"UINT16 \"/>");
95:         req_reg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
96:         req_reg.destination.setPort(my_port);
97:         req_reg.id = DATA_PROVIDER;
98:         req_reg.Send();
99:     }
100:     else
101:     {
102:         if(data_provider.getSensorID() == 0)
103:         {
104:             data_provider = info.source;
105:             data_msg = info.msg_id;
106:             data_manipulator.setMsgDef(info.msg_def);
107:             printf("New Data provider found");
108:             printf(" \n \n%s \n",info.msg_def);
109:             //Send the consume message
110:             consume.Send();
111:         }
112:         else
113:         {
114:             printf("Data provider found");
115:             printf(" - not used \n \n%s \n",info.msg_def);
116:         }
117:     }
118:     break;
119: case SERVICE_PROVIDER:
120:     if(info.type == SDM_REGINFO_CANCELLATION)
121:     {

```

```

122:     service_provider.setSensorID(0);
123:     printf("Service provider failed . . . ");
124:     printf("Searching for new provider \n");
125:     //request info on service providers,
126:     //the DM will repost software tasks
127:     //that could satisfy our requirements
128:
129:     //Set var name to convert which we already know
130:     strcpy(req_reg.item_name,"convert");
131:     //We will not be using wildcards
132:     req_reg.qualist[0] = '\0';
133:     req_reg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
134:     req_reg.destination.setPort(my_port);
135:     req_reg.id = SERVICE_PROVIDER;
136:     req_reg.Send();
137: }
138: else
139: {
140:     if(service_provider.getSensorID() == 0)
141:     {
142:         service_provider = info.source;
143:         service_msg = info.msg_id;
144:         service_manipulator.setMsgDef(info.msg_def);
145:         printf("New Service provider found");
146:         printf(" \n \n%s \n",info.msg_def);
147:     }
148:     else
149:     {
150:         printf("Service provider found - ");
151:         printf("not used \n \n%s \n",info.msg_def);
152:     }
153: }
154: break;
155: }
156:
157: }
158:
159: int main(int argc,char** argv)
160: {
161:     MessageManager mm;
162:     SDMDData dat;

```

```

163:   SDMRegInfo info;
164:   SDMReqReg req_reg;
165:   char buf[BUFSIZE];
166:   long length;
167:
168:   //initialize consumer
169:   SDMInit(argc,argv);
170:   my_port = getPort();
171:   if(my_port == SDM_PM_NOT_AVAILABLE)
172:   {
173:       printf("No PM is available to get port from! \n");
174:       return 0;
175:   }
176:   mm.Async_Init(my_port);
177:
178:   printf("Consumer listening on port %ld \n",my_port);
179:
180:   while(1)
181:   {
182:       if (mm.IsReady())
183:       {
184:           SendHeartbeat();
185: #ifdef WIN32
186:           switch(mm.GetMsg(buf,length))
187: #else
188:           switch(mm.GetMessage(buf,length))
189: #endif
190:           {
191:               case SDM_Data:
192:                   dat.Unmarshal(buf,length);
193:                   DataHandler(dat,length);
194:                   break;
195:               case SDM_RegInfo:
196:                   if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)
197:                   {
198:                       RegInfoHandler(info);
199:                   }
200:                   break;
201:               default:
202:                   printf("Unexpected message \n");
203:           }

```

```

204:     }
205:     else
206:     { //check for data and service providers
207:         if(data_provider.getSensorID() == 0)
208:         {
209:             //request info on integer data providers
210:             //Set variable name
211:             strcpy(req_reg.item_name,"data");
212:             //Set the quallist can be empty
213:             strcpy(req_reg.quallist,"< format= \"UINT16 \"/>");
214:             req_reg.reply
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
215:             req_reg.destination.setPort(my_port);
216:             req_reg.id = DATA_PROVIDER;
217:             req_reg.Send();
218:             printf("Searching for new data provider \n");
219:             sleep(2);
220:         }
221:         if(service_provider.getSensorID() == 0)
222:         {
223:             //request info on service providers
224:             //Set var name to convert which we already know
225:             strcpy(req_reg.item_name,"convert");
226:             //We will not be using wildcards
227:             req_reg.quallist[0] = '\0';
228:             req_reg.reply
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
229:             req_reg.destination.setPort(my_port);
230:             req_reg.id = SERVICE_PROVIDER;
231:             req_reg.Send();
232:             printf("Searching for new service provider \n");
233:             sleep(2);
234:         }
235:         usleep(1000);
236:     }
237: }
238: }

```

File: sdm/app/test/ReqRegTest.cpp

```
1: #include "../common/message/SDMmessage.h"
2: #include "../common/message/SDMReqReg.h"
3: #include "../common/UDPcom.h"
4: #include "../common/message_defs.h"
5: #include "../common/MessageManager/MessageManager.h"
6: #include "../common/message/SDMRegInfo.h"
7: #include <stdio.h>
8: #include <unistd.h>
9: #include <stdlib.h>
10: #include <string.h>
11: #include <signal.h>
12:
13: int sigcount = 0;
14: long my_port = 4030;
15:
16: void SigHandler(int signum);
17: int main(int argc, char** argv)
18: {
19:     SDMInit(argc,argv);
20:     MessageManager mm;
21:     mm.Async_Init(my_port);
22:     signal(SIGINT,SigHandler);
23:     SDMReqReg request;
24:     SDMRegInfo info;
25:     char buf[BUFSIZE];
26:     long length;
27:     request.destination.setPort(my_port);
28:     request.id = 1;
29:     request.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
30:     printf("Sending ReqReg for Message_Count. \n");
31:     strcpy(request.item_name,"Message_Count");
32:     strcpy(request.quallist,"</>");
33:     request.Send();
34:     printf("Sending ReqReg for Enable_Logging. \n");
35:     strcpy(request.item_name,"Enable_Logging");
36:     request.Send();
37:     printf("Sending ReqReg for faults. \n");
38:     strcpy(request.item_name,"faultdata");
39:     request.Send();
```

```

40:
41: while(1)
42: {
43:     if (mm.IsReady())
44:     {
45:         mm.GetMessage(buf,length);
46:         if (info.Unmarshal(buf) > 0)
47:         {
48:             printf("RegInfo received \n%s \n \n",info.msg_def);
49:         }
50:     }
51:     else
52:     {
53:         printf(".");
54:         sleep(1);
55:     }
56: }
57:
58:
59: request.reply = SDM_REQREG_CANCEL;
60: request.Send();
61: return 1;
62: }
63: void SigHandler(int signum)
64: {
65:     SDMReqReg req;
66:     req.reply = SDM_REQREG_CANCEL;
67:     req.destination.setPort(my_port);
68:     if (sigcount == 0)
69:     {
70:         printf("Cancelling ReqReg subscription for Message_Count. \n");
71:         strcpy(req.item_name,"Message_Count");
72:         strcpy(req.quallist,"</>");
73:         req.Send();
74:         sigcount++;
75:     }
76:     else if (sigcount == 1)
77:     {
78:         printf("Cancelling all ReqReg subscriptions. \n");
79:         req.Send();
80:         sigcount++;

```

```
81: }  
82: else  
83:     exit(0);  
84: }
```


File: sdm/app/test/MessageLogRemoveAll.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMCommand.h"
7: #include "../common/MessageManipulator/MessageManipulator.h"
8: #include "../common/MessageManager/MessageManager.h"
9:
10: #include <string.h>
11: #include <unistd.h>
12: #include <stdio.h>
13: #include <sys/types.h>
14: #include <sys/wait.h>
15: #include <signal.h>
16:
17: #define DATA_PROVIDER    1
18:
19: long my_port;
20: SDMComponent_ID data_provider;
21: SDMComponent_ID service_provider;
22: unsigned char service_msg = 0;
23: char msg_types[23] = "qruvtabcdilnomxyzABCDH";
24:
25: MessageManipulator data_manipulator;
26:
27: void RegInfoHandler(SDMRegInfo& info)
28: {
29:     SDMCommand command;
30:     SDMReqReg req_reg;
31:     char type;
32:
33:     //Set the port we will be receiving on
34:     command.destination.setPort(my_port);
35:     //copy the sensor id into the consume message
36:     command.source=info.source;
37:     //copy the msg id into the consume message
38:     command.command_id=info.msg_id;
39:     if (info.id != DATA_PROVIDER)
```

```

40:     return;
41:
42: if (info.source.getAddress() == TaskManager.getAddress() && info.source.getPort() ==
TaskManager.getPort())
43: {
44:     printf("Send log command to TM? (y/n) ");
45:     type = getchar();
46:     getchar();
47:     if (type == 'n' || type == 'N')
48:         return;
49: }
50: else if (info.source.getAddress() == DataManager.getAddress() && info.source.getPort() ==
DataManager.getPort())
51: {
52:     printf("Send log command to DM? (y/n) ");
53:     type = getchar();
54:     getchar();
55:     if (type == 'n' || type == 'N')
56:         return;
57: }
58: else if (info.source.getPort() == PORT_SM)
59: {
60:     printf("Send log command to SM? (y/n) ");
61:     type = getchar();
62:     getchar();
63:     if (type == 'n' || type == 'N')
64:         return;
65: }
66: else if (info.source.getPort() == PORT_PM)
67: {
68:     printf("Send log command to PM? (y/n) ");
69:     type = getchar();
70:     getchar();
71:     if (type == 'n' || type == 'N')
72:         return;
73: }
74:
75: data_provider = info.source;
76: data_manipulator.setMsgDef(info.msg_def);
77: command.length = 1;
78: printf(" \n \n%s \n",info.msg_def);

```

```

79: for (int i = 0; i < 22; i++)
80: {
81:     printf("Sending remove logging command for message type %c. \n",msg_types[i]);
82:     PUT_UCHAR(&command.data[0], msg_types[i]);
83:     command.Send();
84:     usleep(100);
85:
86: }
87: }
88:
89: int main(int argc,char** argv)
90: {
91:     MessageManager mm;
92:     SDMDData dat;
93:     SDMRegInfo info;
94:     SDMReqReg req_reg;
95:     char buf[BUFSIZE];
96:     long length;
97:     bool infosDone = false;
98:     int numRecd = 0;
99:
100:    //initialize consumer
101:    SDMInit(argc,argv);
102:    my_port = getPort();
103:    mm.Async_Init(my_port);
104:
105:    printf("Consumer listening on port %ld \n",my_port);
106:
107:    while(!infosDone)
108:    {
109:        if (mm.IsReady())
110:        {
111:            switch(mm.GetMessage(buf,length))
112:            {
113:                case SDM_RegInfo:
114:                    if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)
115:                    {
116:                        numRecd++;
117:                        RegInfoHandler(info);
118:                    }
119:                    else

```

```

120:         {
121:             if (numRecd == 0)
122:             {
123:                 printf("No log messages found. \n");
124:                 fflush(NULL);
125:             }
126:             infosDone = true;
127:         }
128:         break;
129:     default:
130:         printf("Unexpected message \n");
131:     }
132: }
133: else
134: { //check for data and service providers
135:     if(data_provider.getSensorID() == 0)
136:     {
137:         //request info on integer data providers
138:         //Set variable name
139:         strcpy(req_reg.item_name,"Disable_Logging");
140:         //Set the quallist can be empty
141:         strcpy(req_reg.quallist,"< >");
142:         req_reg.reply = SDM_REQREG_CURRENT_AND_FUTURE;
143:         req_reg.destination.setPort(my_port);
144:         req_reg.id = DATA_PROVIDER;
145:         req_reg.Send();
146:         printf("Searching for log command... \n");
147:         sleep(2);
148:     }
149:     usleep(1000);
150: }
151: }
152: }

```

File: sdm/app/test/ReqxTEDStest.cpp

```
1: #include "../common/message/SDMmessage.h"
2: #include "../common/message/SDMReqxTEDS.h"
3: #include "../common/message/SDMxTEDSInfo.h"
4: #include "../common/MessageManager/MessageManager.h"
5: #include "../common/UDPcom.h"
6:
7: #include <stdio.h>
8: #include <unistd.h>
9: #include <string.h>
10: #include <sys/socket.h>
11: #include <netinet/in.h>
12: #include <arpa/inet.h>
13:
14: void ReplyTest(void);
15: void ReplyOnPortTest(void);
16: void BadNameTest(void);
17: void DMPortTest(void);
18:
19: int main(int argc, char** argv)
20: {
21:
22:     SDMInit(argc,argv);
23:     BadNameTest();
24:     DMPortTest();
25:     ReplyOnPortTest();
26: }
27:
28: void ReplyOnPortTest(void)
29: {
30:     SDMReqxTEDS request;
31:     SDMxTEDSInfo info;
32:     char buf[BUFSIZE];
33:     long port = 4000;
34:     MessageManager mm;
35:     long length;
36:
37:     mm.Async_Init(port);
38:
39:     request.select = 2;
```

```

40: request.source.setSensorID(1);
41: strcpy(request.device_name,"DataManager");
42: request.destination.setPort(port);
43: printf("Starting reply on port test \n");
44: request.Send();
45: if (mm.BlockGetMessage(buf, length) == SDM_xTEDSInfo)
46: {
47:     info.Unmarshal(buf);
48:     printf("Command Byte:%c id:%lu datalength:%ld",buf[0],info.source.getSensorID(),length);
49:     printf(" \n%s \n",info.xTEDS);
50:     fflush(NULL);
51: }
52:
53: sleep(1);
54:
55: request.select = 3;
56: request.Send();
57: if (mm.BlockGetMessage(buf,length) == SDM_xTEDSInfo)
58: {
59:     info.Unmarshal(buf);
60:     printf("Command Byte:%c id:%lu datalength:%ld",buf[0],info.source.getSensorID(),length);
61:     printf(" \n%s \n",info.xTEDS);
62:     fflush(NULL);
63: }
64:
65: printf("Reply on port test done \n");
66: fflush(NULL);
67: sleep(5);
68: }
69:
70: void BadNameTest(void)
71: {
72: char msg[100];
73: memset(msg,0,100);
74: msg[0] = SDM_ReqxTEDS;
75: msg[5] = 1;
76: printf("Starting Bad Name Test \n");
77: struct in_addr inaddr;
78: inaddr.s_addr = DataManager.getAddress();
79: UDPsendto(inet_ntoa(inaddr),PORT_DM,msg,100);
80: printf("Bad Name Test done \n");

```

```
81: fflush(NULL);
82: sleep(5);
83: }
84:
85: void DMPortTest(void)
86: {
87:   SDMReqxTEDS request;
88:
89:   request.select = 2;
90:   request.source.setSensorID(1);
91:   strcpy(request.device_name,"DataManager");
92:   request.destination.setPort(PORT_DM);
93:   printf("Starting DM port test \n");
94:   request.Send();
95:   request.select = 3;
96:   request.Send();
97:   printf("DM port test done \n");
98:   fflush(NULL);
99:   sleep(5);
100: }
```

File: sdm/app/test/README

1: SDM/app/test folder

2:

3: This folder and its subfolders contain ongoing test applications for the SDM system.

4: Documentation about these tests is mostly non-existent. This folder will be provided

5: with releases for those who may be interested.

6:

7: The SDM/app/test/ folder contains generic tests for the SDM system.

8:

9: The SDM/app/test/ClassTests/ folder contains tests for SDM classes.

10:

11: The SDM/app/test/DMTests/ folder contains tests specific to the Data Manager.

12:

13:

14:

15: Most of these applications should be started similarly to how the Process Manager would

16: start-up applications. That is, with the same command-line string.

17: \$./APPNAME TM-Address DM-Address SDM-PID

18: For example:

19: \$./SearchTest 127.0.0.1 127.0.0.1 56

File: sdm/app/test/magnetometer_test.cpp

```
1: #include "../common/message/SDMmessage.h"
2: #include "../common/message/SDMData.h"
3: #include "../common/message/SDMReqReg.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMConsume.h"
6: #include "../common/MessageManager/MessageManager.h"
7:
8: #include <stdio.h>
9: #include <unistd.h>
10: #include <string.h>
11:
12: int main(int argc, char** argv)
13: {
14:     SDMInit(argc, argv);
15:     SDMReqReg request;
16:     SDMRegInfo info;
17:     SDMData data;
18:     SDMConsume consume;
19:     MessageManager mm;
20:     char buf[BUFSIZE];
21:
22:     int result;
23:     int my_port = getPort();
24:     if (my_port < 0)
25:     {
26:         printf("Error getting port. \n");
27:         return -1;
28:     }
29:     mm.Async_Init(my_port);
30:     double timestamp;
31:     float Field_X;
32:     float Field_Y;
33:     float Field_Z;
34:
35:     strncpy(request.item_name, "MagFieldData", 81);
36:     strcpy(request.quallist, "<description= \"TFM100S Magnetometer Fields \"/>");
37:     request.destination.setPort(my_port);
38:     request.Send();
39:     result = info.Recv(my_port);
```

```

40: while(result == SDM_NO_FURTHER_DATA_PROVIDER)
41: {
42:     sleep(1);
43:     printf("Searching for Magnetometer... \n");
44:     request.Send();
45:     result = info.Recv(my_port);
46: }
47: consume.source = info.source;
48: consume.msg_id = info.msg_id;
49: consume.destination.setPort(my_port);
50: while(result != SDM_NO_FURTHER_DATA_PROVIDER)
51: {
52:     result = info.Recv(my_port);
53: }
54: printf("Magnetometer found, requesting field data \n");
55: consume.Send();
56: while(1)
57: {
58:     if (mm.BlockGetMessage(buf) != SDM_Data)
59:         continue;
60:     result = data.Unmarshal(buf);
61:     if(result == SDM_INVALID_MESSAGE)
62:     {
63:         printf("Invalid Message recieved \n");
64:     }
65:     else
66:     {
67:         //get timestamp (DFP)
68:         memcpy(&timestamp,data.msg,8);
69:         //get Field_X (SFP)
70:         memcpy(&Field_X,data.msg+8,4);
71:         //get Field_Y (SFP)
72:         memcpy(&Field_Y,data.msg+12,4);
73:         //get Field_Z (SFP)
74:         memcpy(&Field_Z,data.msg+16,4);
75:         printf("time:%f \tX:%f \tY:%f \tZ:%f \n",timestamp,Field_X,Field_Y,Field_Z);
76:     }
77: }
78: }

```

File: sdm/app/test/RoboHubTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <signal.h>
4: #include <unistd.h>
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMCommand.h"
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/MessageManager/MessageManager.h"
9: #include "../common/message/SDMData.h"
10: #include "../common/message/SDMConsume.h"
11: #include "../common/message/SDMCancel.h"
12:
13: #define RH_STATUS    1
14: #define RH_TEMPREADING  2
15:
16: SDMComponent_ID sub_compid;
17: SDMMessage_ID sub_msgid;
18: SDMMessage_ID StatusID;
19: SDMMessage_ID TempReadingID;
20: const int myPort = 4058;
21: void SigHandler(int signum);
22:
23: int main (int argc, char ** argv)
24: {
25:     MessageManager mm;
26:     SDMReqReg reqreg_msg;
27:     char buf[BUFSIZE];
28:     SDMRegInfo reginfo_msg;
29:     SDMCommand cmd_msg;
30:     SDMData data_msg;
31:     SDMConsume cons_msg;
32:     unsigned char type;
33:     SDMInit(argc, argv);
34:     signal(SIGINT,SigHandler);
35:
36:     mm.Async_Init(myPort);
37:
38:     printf(" \nSending ReqReg for robohub status message... \n");
39:     strcpy(reqreg_msg.device,"RoboHub_8_Port");
```

```

40: strcpy(reqreg_msg.item_name,"GetHubStatus");
41: reqreg_msg.id = RH_STATUS;
42: reqreg_msg.destination.setPort(myPort);
43: reqreg_msg.Send();
44:
45: printf("Sending ReqReg for robohub temperature reading... \n");
46: strcpy(reqreg_msg.item_name,"DeviceTemp");
47: reqreg_msg.id = RH_TEMPREADING;
48: reqreg_msg.Send();
49:
50: while (1)
51: {
52:     type = mm.BlockGetMessage(buf);
53:     switch (type)
54:     {
55:         case SDM_RegInfo:
56:             if (reginfo_msg.Unmarshal(buf) != SDM_NO_FURTHER_DATA_PROVIDER)
57:             {
58:                 //Robohub status message reginfo
59:                 if (reginfo_msg.id == RH_STATUS)
60:                 {
61:                     cmd_msg.source = reginfo_msg.source;
62:                     cmd_msg.command_id = reginfo_msg.msg_id;
63:                     cmd_msg.length = 0;
64:                     printf("Sending command message to get RoboHub status. \n");
65:                     cmd_msg.Send(reginfo_msg.source);
66:                 }
67:                 //Robohub temperature reading message
68:                 else if (reginfo_msg.id == RH_TEMPREADING)
69:                 {
70:                     cons_msg.source = reginfo_msg.source;
71:                     sub_comp_id = reginfo_msg.source;
72:                     cons_msg.destination.setPort(myPort);
73:                     TempReadingID = cons_msg.msg_id = reginfo_msg.msg_id;
74:                     sub_msg_id = reginfo_msg.msg_id;
75:                     printf("Sending consume for robohub temperature reading... \n");
76:                     cons_msg.Send();
77:                 }
78:             }
79:             break;
80:             case SDM_Data:

```

```

81:         if (data_msg.Unmarshal(buf) < 0)
82:         {
83:             printf("Invalid data message received. \n");
84:             continue;
85:         }
86:         //If this is the temperature reading response
87:         if (data_msg.msg_id == TempReadingID)
88:         {
89:             //pull out the temperature measurement
90:             short TempVal = GET_SHORT(data_msg.msg+8);
91:             printf("Temperature reading received: %hd \n",TempVal);
92:         }
93:         //This is the status command reply
94:         else
95:         {
96:             //pull out the port enumeration
97:             unsigned char PortEnum = data_msg.msg[8];
98:             printf("Received port enumeration: %hhu \n",PortEnum);
99:         }
100:         break;
101:     default:
102:         printf("Unexpected message (%d). \n",type);
103:     }
104: }
105:
106:
107: }
108:
109: void SigHandler(int signum)
110: {
111:     SDMCancel cancel_msg;
112:     cancel_msg.source = sub_compid;
113:     cancel_msg.msg_id = sub_msgid;
114:     cancel_msg.destination.setPort(myPort);
115:     cancel_msg.Send();
116:     _exit(0);
117: }

```

File: sdm/app/test/SearchSubTest.cpp

```
1: #include "../common/message/SDMSearch.h"
2: #include "../common/message/SDMSearchReply.h"
3: #include "../common/MessageManager/MessageManager.h"
4: #include <string.h>
5: #include <sys/types.h>
6: #include <sys/stat.h>
7: #include <fcntl.h>
8: #include <unistd.h>
9: #include <stdio.h>
10: #include <stdlib.h>
11: #include <pthread.h>
12: #include <signal.h>
13:
14: #define INTERFACE_SEARCH_ID 1
15: #define DEVICE_SEARCH_ID 2
16: #define APPLICATION_SEARCH_ID 3
17:
18: long my_port = 4020;
19: int sigint_count = 0;
20:
21: void SigHandler(int signum);
22: int main(int argc, char** argv)
23: {
24:     SDMSearch s;
25:     SDMSearchReply reply;
26:     int result = 0;
27:     char buf[BUFSIZE];
28:     long length;
29:     MessageManager mm;
30:     int cur = 0;
31:     int count = 0;
32:
33:     signal(SIGINT, SigHandler);
34:     SDMInit(argc, argv);
35:     my_port = getPort();
36:     if(my_port == SDM_PM_NOT_AVAILABLE)
37:     {
38:         printf("Unable to get a port from the Process Manager! \n");
39:         exit(0);
```

```

40: }
41: mm.Async_Init(my_port);
42: sleep(1);
43: s.destination.setPort(my_port);
44: s.destination.setAddress(DataManager.getAddress()); //This application must be run on the same
node as the DM
45: s.id = INTERFACE_SEARCH_ID;
46: s.reply = SDM_SEARCH_CURRENT_AND_FUTURE;
47:
48: strcpy(s.reg_expr, ".*?<Interface name= \"(.*)\".*?>.*?<Interface name= \"(.*)\".*?>(.*?<Interface
name= \"(.*)\".*?>.*?)*|.??<Interface name= \"(.*)\".*?>.*?<Interface name= \"(.*)\"
.*?>.*?|.??<Interface name= \"(.*)\".*?>.*?");
49:
50: s.Send();
51:
52: printf(" \nSending SDMSearch for registered Devices \n");
53: memset(s.reg_expr, 0, 512);
54: strcpy(s.reg_expr, ".*?<Device.*?name= \"(.*)\".*?");
55: s.id = DEVICE_SEARCH_ID;
56: s.Send();
57:
58: printf("Sending SDMSearch for registered Applications \n");
59: memset(s.reg_expr, 0, 512);
60: strcpy(s.reg_expr, ".*?<Application.*?name= \"(.*)\".*?");
61: s.id = APPLICATION_SEARCH_ID;
62: s.Send();
63: while(1)
64: {
65:     if (mm.IsReady())
66:     {
67:         mm.GetMessage(buf, length);
68:         cur = 0;
69:         result = reply.Unmarshal(buf);
70:         if(result == SDM_NO_FURTHER_DATA_PROVIDER)
71:             continue;
72:         printf(" \n");
73:         if (reply.id == INTERFACE_SEARCH_ID)
74:             printf("Reply for interface search: \n");
75:         else if (reply.id == DEVICE_SEARCH_ID)
76:             printf("Reply for device search: \n");
77:         else if (reply.id == APPLICATION_SEARCH_ID)
78:             printf("Reply for application search: \n");

```

```

79:         if(count != 0)
80:             printf(" \n");
81:             printf("Match(es)                from                %lu:%lu:%d
\n",reply.source.getSensorID(),reply.source.getAddress(),reply.source.getPort());
82:             count++;
83:             while(reply.captured_matches[cur]!=0)
84:             {
85:                 if(strlen(reply.captured_matches+cur)<32)
86:                     printf("Match is %s \n",reply.captured_matches+cur);
87:                 else
88:                     printf("Match length of %d exceeds name length restriction of 32
\n",strlen(reply.captured_matches+cur));
89:                 cur += strlen(reply.captured_matches+cur)+1;
90:             }
91:
92:         }
93:     else
94:     {
95:         printf(".");
96:         sleep(1);
97:         fflush(NULL);
98:     }
99: }
100: }
101: void SigHandler(int signum)
102: {
103:     SDMSearch search;
104:     if (sigint_count == 0)
105:     {
106:         printf("Sending search subscription cancellation for interface searches only. \n");
107:         search.reply = SDM_SEARCH_CANCEL;
108:         search.destination.setPort(my_port);
109:         strcpy(search.reg_expr,".*?<Interface name= \"(.*)\" \".*?>.*?<Interface name= \"(.*)
\n.*?>(.?<Interface name= \"(.*)\" \".*?>.*?)*|.??<Interface name= \"(.*)\" \".*?>.*?<Interface name=
\n(.*)\" \".*?>.*?|.??<Interface name= \"(.*)\" \".*?>.*?");
110:         search.Send();
111:         sigint_count++;
112:     }
113:     else if (sigint_count == 1)
114:     {
115:         printf("Sending search subscription cancellations. \n");
116:         search.reply = SDM_SEARCH_CANCEL;

```



```
117:     search.destination.setPort(my_port);
118:     search.Send();
119:     sigint_count++;
120: }
121: else
122:     exit(0);
123: }
124:
```

File: sdm/app/test/MessageLogAddAll.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMCommand.h"
7: #include "../common/MessageManipulator/MessageManipulator.h"
8: #include "../common/MessageManager/MessageManager.h"
9:
10: #include <string.h>
11: #include <unistd.h>
12: #include <stdio.h>
13: #include <sys/types.h>
14: #include <sys/wait.h>
15: #include <signal.h>
16:
17: #define DATA_PROVIDER    1
18:
19: long my_port;
20: SDMComponent_ID data_provider;
21: SDMComponent_ID service_provider;
22: unsigned char service_msg = 0;
23: char msg_types[23] = "qruvtabcdilnomxyzABCDH";
24:
25: MessageManipulator data_manipulator;
26:
27: void RegInfoHandler(SDMRegInfo& info)
28: {
29:     SDMCommand command;
30:     SDMReqReg req_reg;
31:     char type;
32:
33:     //Set the port we will be receiving on
34:     command.destination.setPort(my_port);
35:     //copy the sensor id into the consume message
36:     command.source=info.source;
37:     //copy the msg id into the consume message
38:     command.command_id=info.msg_id;
39:     if (info.id != DATA_PROVIDER)
```

```

40:     return;
41:
42: if (info.source.getPort() == PORT_TM)
43: {
44:     printf("Send log command to TM? (y/n) ");
45:     type = getchar();
46:     getchar();
47:     if (type == 'n' || type == 'N')
48:         return;
49: }
50: else if (info.source.getPort() == PORT_DM)
51: {
52:     printf("Send log command to DM? (y/n) ");
53:     type = getchar();
54:     getchar();
55:     if (type == 'n' || type == 'N')
56:         return;
57: }
58: else if (info.source.getPort() == PORT_SM)
59: {
60:     printf("Send log command to SM? (y/n) ");
61:     type = getchar();
62:     getchar();
63:     if (type == 'n' || type == 'N')
64:         return;
65: }
66: else if (info.source.getPort() == PORT_PM)
67: {
68:     printf("Send log command to PM? (y/n) ");
69:     type = getchar();
70:     getchar();
71:     if (type == 'n' || type == 'N')
72:         return;
73: }
74:
75: data_provider = info.source;
76: data_manipulator.setMsgDef(info.msg_def);
77: command.length = 1;
78: for (int i = 0; i < 22; i++)
79: {
80:     PUT_UCHAR(&command.data[0], msg_types[i]);

```

```

81:     usleep(100);
82:     printf("Enable logging command sent for message type %c. \n",msg_types[i]);
83:     command.Send();
84: }
85: }
86:
87: int main(int argc,char*** argv)
88: {
89:     MessageManager mm;
90:     SDMDData dat;
91:     SDMRegInfo info;
92:     SDMReqReg req_reg;
93:     char buf[BUFSIZE];
94:     long length;
95:     bool infosDone = false;
96:     int numRecd = 0;
97:
98:     //initialize consumer
99:     SDMInit(argc,argv);
100:     my_port = getPort();
101:     mm.Async_Init(my_port);
102:
103:     printf("Consumer listening on port %ld \n",my_port);
104:
105:     while(!infosDone)
106:     {
107:         if (mm.IsReady())
108:         {
109:             switch(mm.GetMessage(buf,length))
110:             {
111:                 case SDM_RegInfo:
112:                     if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)
113:                     {
114:                         numRecd++;
115:                         RegInfoHandler(info);
116:                     }
117:                 else
118:                 {
119:                     if (numRecd == 0)
120:                     {
121:                         printf("No log messages found. \n");

```

```

122:             fflush(NULL);
123:         }
124:         infosDone = true;
125:     }
126:     break;
127: default:
128:     printf("Unexpected message \n");
129: }
130: }
131: else
132: { //check for data and service providers
133:     if(data_provider.getSensorID() == 0)
134:     {
135:         //request info on integer data providers
136:         //Set variable name
137:         strcpy(req_reg.item_name,"Enable_Logging");
138:         //Set the quallist can be empty
139:         strcpy(req_reg.quallist,"<>");
140:         req_reg.reply = SDM_REQREG_CURRENT_AND_FUTURE;
141:         req_reg.destination.setPort(my_port);
142:         req_reg.id = DATA_PROVIDER;
143:         req_reg.Send();
144:         printf("Searching for log command... \n");
145:         sleep(2);
146:     }
147:     usleep(1000);
148: }
149: }
150: }

```

File: sdm/app/test/TatTest.cpp

```
1: #include "../common/message/SDMTat.h"
2:
3: int main (int argc, char** argv)
4: {
5:     SDMInit(argc, argv);
6:
7:     SDMTat Tat;
8:
9:     Tat.destination.setSensorID(0);
10:    Tat.seconds = 1;
11:    Tat.useconds = 2;
12:
13:    Tat.Send();
14:
15:    return 0;
16: }
```

File: sdm/app/test/TMSubsTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <unistd.h>
4: #include <vector>
5: #include <signal.h>
6:
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/message/SDMReqReg.h"
9: #include "../common/message/SDMConsume.h"
10: #include "../common/message/SDMCancel.h"
11: #include "../common/message/SDMData.h"
12: #include "../common/message/SDMCommand.h"
13: #include "../common/message/SDMService.h"
14: #include "../common/message/SDMSerreqst.h"
15: #include "../common/message/SDMMessage_ID.h"
16: #include "../common/message/SDMCommand.h"
17: #include "../common/MessageManager/MessageManager.h"
18:
19: #define TMSUB_STATUS      1
20: #define TMSUB_TASKQUEUED  2
21: #define TMSUB_TASKSTARTED 3
22: #define TMSUB_TASKFINISHED 4
23: #define TMSER_NAMETOPID   5
24: #define TMDAT_NAMETOPID   8
25: #define TMCMD_KILLTASK    6
26: #define TMCMD_STARTTASK   7
27: #define TMSER_TASKLIST    10
28: #define TMSER_TASKLISTREPLY 11
29: #define TMSER_RUNNINGTASKLIST 9
30: #define TMSER_RUNNINGTASKLISTREPLY 12
31: #define TMSER_TASKINFO    13
32: #define TMSER_TASKINFOREPLY 14
33:
34: using namespace std;
35:
36: SDMMessage_ID status_id;
37: SDMMessage_ID queued_id;
38: SDMMessage_ID started_id;
39: SDMMessage_ID finished_id;
```

```

40: SDMMMessage_ID nametopid_id;
41: SDMMMessage_ID nametopidreply_id;
42: SDMMMessage_ID starttask_id;
43: SDMMMessage_ID killtask_id;
44: SDMMMessage_ID tasklist_id;
45: SDMMMessage_ID tasklistreply_id;
46: SDMMMessage_ID runningtasklist_id;
47: SDMMMessage_ID runningtasklistreply_id;
48: SDMMMessage_ID taskinfo_id;
49: SDMMMessage_ID taskinforeply_id;
50:
51: const int my_port = 4999;
52: bool providers_found = false;
53: vector<long> sensor_ids;
54: vector<SDMComponent_ID> comp_ids;
55: SDMComponent_ID TMID;
56: int signals_recd;
57: bool convert_name_ready = false;
58: const char* START_TASKNAME = "messagecountconsumer";
59: //Tests whether the given argument is contained within sensor_ids
60: bool sensor_ids_contains(long num)
61: {
62: for (unsigned int i = 0; i < sensor_ids.size(); i++)
63: {
64:     if (sensor_ids[i] == num)
65:         return true;
66: }
67: return false;
68: }
69: bool comp_ids_contains(SDMComponent_ID id)
70: {
71: for (unsigned int i = 0; i < comp_ids.size(); i++)
72: {
73:     if (comp_ids[i] == id)
74:         return true;
75: }
76: return false;
77: }
78:
79:
80: void RegInfoHandler(SDMRegInfo reg)

```



```

81: {
82:   SDMConsume consume;
83:   consume.destination.setPort(my_port);
84:   consume.source = reg.source;
85:   consume.msg_id = reg.msg_id;
86:
87:   switch(reg.id)
88:   {
89:     case TMSUB_STATUS:
90:       printf("Task manager's Status subscription found, subscribing... \n");
91:       status_id = reg.msg_id;
92:       TMID = reg.source;
93:       consume.Send();
94:       providers_found = true;
95:       break;
96:     case TMSUB_TASKQUEUED:
97:       printf("Task manager's TaskQueued subscription found, subscribing... \n");
98:       queued_id = reg.msg_id;
99:       TMID = reg.source;
100:      consume.Send();
101:      providers_found = true;
102:      break;
103:     case TMSUB_TASKSTARTED:
104:       printf("Task manager's TaskStarted subscription found, subscribing... \n");
105:       started_id = reg.msg_id;
106:       TMID = reg.source;
107:       consume.Send();
108:       providers_found = true;
109:       break;
110:     case TMSUB_TASKFINISHED:
111:       printf("Task manager's TaskFinished subscription found, subscribing... \n");
112:       finished_id = reg.msg_id;
113:       TMID = reg.source;
114:       consume.Send();
115:       providers_found = true;
116:       break;
117:     case TMSER_NAMETOPID:
118:       printf("Task manager's NameToPID subscription found, subscribing... \n");
119:       nametopid_id = reg.msg_id;
120:       TMID = reg.source;
121:       providers_found = true;

```

```

122:         break;
123:     case TMDAT_NAMETOPID:
124:         nametopidreply_id = reg.msg_id;
125:         break;
126:     case TMCMD_STARTTASK:
127:         printf("Task manager's StartTask subscription found, subscribing... \n");
128:         starttask_id = reg.msg_id;
129:         break;
130:     case TMCMD_KILLTASK:
131:         printf("Task manager's KillTask subscription found, subscribing... \n");
132:         killtask_id = reg.msg_id;
133:         break;
134:     case TMSER_TASKLIST:
135:         printf("Found task list request. \n");
136:         tasklist_id = reg.msg_id;
137:         break;
138:     case TMSER_TASKLISTREPLY:
139:         tasklistreply_id = reg.msg_id;
140:         break;
141:     case TMSER_RUNNINGTASKLISTREPLY:
142:         runningtasklistreply_id = reg.msg_id;
143:         break;
144:     case TMSER_RUNNINGTASKLIST:
145:         printf("Found running task list request. \n");
146:         runningtasklist_id = reg.msg_id;
147:         break;
148:     case TMSER_TASKINFO:
149:         printf("Found task info request. \n");
150:         taskinfo_id = reg.msg_id;
151:         break;
152:     case TMSER_TASKINFOREPLY:
153:         taskinforeply_id = reg.msg_id;
154:         break;
155:     default:
156:         printf("Received unexpected Reg Info type. \n");
157:     }
158: }
159:
160: void SendRegReqs()
161: {
162:     SDMReqReg request;

```

```

163: request.destination.setPort(my_port);
164: strcpy(request.interface, "TM_Interface");
165:
166: printf("Sending request for the Status subscription. \n");
167: request.id = TMSUB_STATUS;
168: strcpy(request.item_name, "Status");
169: request.Send();
170:
171: printf("Sending request for the TaskQueued subscription. \n");
172: request.id = TMSUB_TASKQUEUED;
173: strcpy(request.item_name, "TaskQueued");
174: request.Send();
175:
176: printf("Sending request for the TaskStarted subscription. \n");
177: request.id = TMSUB_TASKSTARTED;
178: strcpy(request.item_name, "TaskStarted");
179: request.Send();
180:
181: printf("Sending request for TaskFinshed subscription. \n \n");
182: strcpy(request.item_name, "TaskFinished");
183: request.id = TMSUB_TASKFINISHED;
184: request.Send();
185:
186: printf("Sending request for Task name to PID service. \n \n");
187: strcpy(request.item_name, "NameToPID");
188: request.id = TMSER_NAMETOPID;
189: request.Send();
190:
191: strcpy(request.item_name, "TaskPID");
192: request.id = TMDAT_NAMETOPID;
193: request.Send();
194:
195: printf("Sending request for start task command. \n \n");
196: strcpy(request.item_name, "StartTask");
197: request.id = TMCMD_STARTTASK;
198: request.Send();
199:
200: printf("Sending request for kill task command. \n \n");
201: strcpy(request.item_name, "KillTask");
202: request.id = TMCMD_KILLTASK;
203: request.Send();

```

```

204:
205:     printf("Sending request for task list command. \n \n");
206:     strcpy(request.item_name, "GetTaskList");
207:     request.id = TMSE_TASKLIST;
208:     request.Send();
209:
210:     printf("Sending request for running task list command. \n \n");
211:     strcpy(request.item_name, "GetRunningTaskList");
212:     request.id = TMSE_RUNNINGTASKLIST;
213:     request.Send();
214:
215:     printf("Sending request for running task list command. \n \n");
216:     strcpy(request.item_name, "RunningTaskListReply");
217:     request.id = TMSE_RUNNINGTASKLISTREPLY;
218:     request.Send();
219:
220:     printf("Sending request for running task list command. \n \n");
221:     strcpy(request.item_name, "TaskListReply");
222:     request.id = TMSE_TASKLISTREPLY;
223:     request.Send();
224:
225:     printf("Sending request for getting task info. \n \n");
226:     strcpy(request.item_name, "GetTaskInfo");
227:     request.id = TMSE_TASKINFO;
228:     request.Send();
229:
230:     strcpy(request.item_name, "TaskInfoReply");
231:     request.id = TMSE_TASKINFOREPLY;
232:     request.Send();
233: }
234:
235: void DataHandler(SDMDData data)
236: {
237:     unsigned char mode;
238:     int exit_status;
239:     char task_name[MAX_FILENAME_SIZE];
240:     SDMSservice request;
241:     bool PidRequest = false;
242:     printf("Data message received. \n");
243:     if (data.msg_id == status_id)
244:     {

```

```

245:     mode = GET_UCHAR(data.msg);
246:     printf("Status message received mode is %hd. \n", mode);
247: }
248: else if (data.msg_id == queued_id)
249: {
250:     strncpy(task_name, data.msg, MAX_FILENAME_SIZE);
251:     printf("TaskQueued message received for task \"%s\". \n", task_name);
252:     //PidRequest = true;
253:     SDMSservice msgService;
254:     msgService.command_id = tasklist_id;
255:     msgService.source = TMID;
256:     msgService.destination.setPort(my_port);
257:     msgService.Send();
258:     printf("Sending task list request... \n");
259: }
260: else if (data.msg_id == started_id)
261: {
262:     strncpy(task_name, data.msg, MAX_FILENAME_SIZE);
263:     unsigned int uiPid = GET_UINT(data.msg + MAX_FILENAME_SIZE);
264:     printf("TaskStarted message received for task \"%s\", pid %u \n", task_name, uiPid);
265:     if (strcmp(task_name, START_TASKNAME) == 0)
266:         PidRequest = true;
267:     SDMSservice msgService;
268:     msgService.command_id = runningtasklist_id;
269:     msgService.source = TMID;
270:     msgService.destination.setPort(my_port);
271:     msgService.Send();
272:     printf("Sending task list request... \n");
273:
274:     printf("Sending task info request... \n");
275:     msgService.command_id = taskinfo_id;
276:     PUT_UINT(msgService.data, uiPid);
277:     msgService.length = sizeof(unsigned int);
278:     msgService.Send();
279: }
280: else if (data.msg_id == finished_id)
281: {
282:     strncpy(task_name, data.msg, MAX_FILENAME_SIZE);
283:     unsigned int uiPid = GET_UINT(data.msg+MAX_FILENAME_SIZE);
284:     exit_status = GET_INT(data.msg+MAX_FILENAME_SIZE + sizeof(unsigned int));

```

```

285:     printf("TaskFinished message received for task  \"%s \", pid %u, and returned %d. \n",
task_name, uiPid, exit_status);
286:
287:     //PidRequest = true;
288: }
289: else if (data.msg_id == nametopidreply_id)
290: {
291:     int PID = GET_INT(data.msg + MAX_FILENAME_SIZE);
292:     printf("Received PID number %d for task %s. \n",PID, data.msg);
293:
294:     printf("Killing task %s \n",data.msg);
295:     SDMCommand cmdMsg;
296:     cmdMsg.command_id = killtask_id;
297:     PUT_INT(cmdMsg.data, PID);
298:     cmdMsg.length = 4;
299:     cmdMsg.SendTo(data.source);
300: }
301: else if (data.msg_id == tasklistreply_id || data.msg_id == runningtasklistreply_id)
302: {
303:     if (data.msg_id == tasklistreply_id)
304:         printf("--- Task list is: \n");
305:     else
306:         printf("--- Running task list is: \n");
307:     printf("%s \n",data.msg);
308:     /* unsigned int uiCurPid;
309:     char strTaskName[512];
310:     unsigned int uiCurOffset = 0;
311:     bool bDoubleNullFound = false;
312:
313:     unsigned short usNumTasks = GET_USHORT(&data.msg[uiCurOffset]);
314:     uiCurOffset += sizeof (unsigned short);
315:
316:     printf (" -- %hu items -- \n", usNumTasks);
317:     //while (!bDoubleNullFound)
318:     for (unsigned short i = 0; i < usNumTasks; i++)
319:     {
320:         uiCurPid = GET_UINT(&data.msg[uiCurOffset]);
321:         uiCurOffset += sizeof(uiCurPid);
322:
323:         strcpy(strTaskName, &data.msg[uiCurOffset]);
324:         uiCurOffset += strlen(strTaskName) + 1;

```

```

325:
326:     printf (" Task \" %s \" pid %u \n", strTaskName, uiCurPid);
327:     */
328:     printf(" ---- \n");
329: }
330: else if (data.msg_id == taskinforeply_id)
331: {
332:     char strTaskName[MAX_FILENAME_SIZE];
333:     unsigned int uiCurBufferOffset = 0;
334:     strcpy(strTaskName, data.msg);
335:     uiCurBufferOffset += MAX_FILENAME_SIZE;
336:
337:     char cArchType = GET_CHAR( &data.msg[uiCurBufferOffset] );
338:     uiCurBufferOffset += sizeof(char);
339:
340:     char cOsType = GET_CHAR(& data.msg[uiCurBufferOffset] );
341:     uiCurBufferOffset += sizeof(char);
342:
343:     char cMemType = GET_CHAR( &data.msg[uiCurBufferOffset] );
344:     uiCurBufferOffset += sizeof(char);
345:
346:     unsigned char ucPmId = GET_UCHAR( &data.msg[uiCurBufferOffset] );
347:     uiCurBufferOffset += sizeof(char);
348:
349:     char cTaskState = GET_CHAR( &data.msg[uiCurBufferOffset] );
350:     uiCurBufferOffset += sizeof(char);
351:
352:     char cExecutionMode = GET_CHAR( &data.msg[uiCurBufferOffset] );
353:     uiCurBufferOffset += sizeof(char);
354:
355:     printf("Task info reply received for %s- \n", strTaskName);
356:     printf(" Arch: %hhd Os: %hhd Mem: %hhd \n", cArchType, cOsType, cMemType);
357:     printf(" PmId: %hhd State: %hhd Mode: %hhd \n", ucPmId, cTaskState, cExecutionMode);
358: }
359: if (PidRequest)
360: {
361:     request.source = TMID;
362:     request.destination.setPort(my_port);
363:     request.command_id = nametopid_id;
364:     strcpy(request.data, data.msg);
365:     request.length = strlen(data.msg);

```

```

366:     request.Send();
367: }
368: }
369:
370: void SigHandler(int sig_num)
371: {
372:     SDMCancel cancel;
373:     cancel.source = comp_ids[0];
374:     cancel.destination.setPort(my_port);
375:     if (sig_num == SIGINT)
376:     {
377:         printf("Canceling subscriptions...");
378:         cancel.msg_id = status_id;
379:         cancel.Send();
380:         cancel.msg_id = queued_id;
381:         cancel.Send();
382:         cancel.msg_id = started_id;
383:         cancel.Send();
384:         cancel.msg_id = finished_id;
385:         cancel.Send();
386:         printf("Done. \n");
387:     }
388:     exit(EXIT_SUCCESS);
389: }
390: int main (int argc, char ** argv)
391: {
392:     char buf[BUFSIZE];
393:     long length;
394:     SDMRegInfo info;
395:     SDMData data;
396:     SDMInit(argc, argv);
397:     int Count = 0;
398:     MessageManager mm;
399:     mm.Async_Init(my_port);
400:     signal(SIGINT, SigHandler);
401:
402:     while (1)
403:     {
404:         if (mm.IsReady())
405:         {
406:             switch(mm.GetMessage(buf, length))

```



```

407:         {
408:             case SDM_RegInfo:
409:                 if (info.Unmarshal(buf) != SDM_NO_FURTHER_DATA_PROVIDER)
410:                     {
411:                         RegInfoHandler(info);
412:                     }
413:                 break;
414:             case SDM_Data:
415:                 data.Unmarshal(buf);
416:                 DataHandler(data);
417:                 break;
418:
419:         }
420:
421:     }
422:     else if (!providers_found)
423:     {
424:         SendRegReqs();
425:         sleep(1);
426:     }
427:     else
428:     {
429:         sleep(1);
430:         /* if (Count++ == 10)
431:         {
432:             printf(" Starting task %s... \n", START_TASKNAME);
433:             SDMCommand request;
434:             request.source = TMID;
435:             request.command_id = starttask_id;
436:             request.destination.setPort(my_port);
437:             const unsigned short ZERO = 0;
438:             PUT_USHORT(request.data, ZERO);
439:             PUT_INT(request.data + 2, 0);
440:             const unsigned int FILENAME_OFFSET = 6;
441:             strcpy(request.data + FILENAME_OFFSET, START_TASKNAME);
442:             request.length = 27;
443:             request.Send();
444:             Count = 0;
445:         }*/
446:     }
447: }

```

```
448:  
449:  
450:    return 0;  
451: }
```

File: sdm/app/test/xTEDSPoster.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMCancelxTEDS.h"
3:
4: #include <stdio.h>
5: #include <stdlib.h>
6: #include <string.h>
7: #include <sys/types.h>
8: #include <sys/stat.h>
9: #include <fcntl.h>
10: #include <unistd.h>
11:
12: long myPort = 0;
13:
14: int main(int argc, char* argv[])
15: {
16:     char xteds[3*BUFSIZE];
17:     int fd = 0;
18:     SDMxTEDS xTEDS;
19:     SDMCancelxTEDS can;
20:     int value = 0;
21:     char cont = 0;
22:     char add[100];
23:     int result = 0;
24:
25:     SDMInit(argc,argv);
26:     myPort = getPort();
27:     if(argc <= 4)
28:     {
29:         printf("./xTEDSPoster TM-ip DM-ip pid xTEDSfileName \n");
30:         return 0;
31:     }
32:     memset(xteds,0,3*BUFSIZE);
33:     fd = open(argv[4],O_RDONLY);
34:     if(fd == -1)
35:     {
36:         perror("File was unable to be opened! \n");
37:         return -1;
38:     }
39:     value = read(fd,xteds,3*BUFSIZE);
```

```

40: xTEDS.source.setAddress(DataManager.getAddress());
41: xTEDS.source.setPort(myPort);
42: xTEDS.source.setSensorID(1);
43: strncpy(xTEDS.xTEDS,xteds,value);
44: printf("Would you like to include a usb address? (y or n): ");
45: cont = getchar();
46: if(cont == 'y')
47: {
48:     printf("SPA Node path (x.x): ");
49:     fflush(NULL);
50:     scanf("%s",add);
51:     strcpy(xTEDS.SPA_node,add);
52:     printf("SPA Node path entered is %s \n",add);
53: }
54: result = xTEDS.Send();
55: if(result < 0)
56: {
57:     switch(result)
58:     {
59:         case -1:
60:             printf("Error: Send Error \n");
61:             return 0;
62:             break;
63:         case -2:
64:             printf("Error: Recv Error \n");
65:             return 0;
66:             break;
67:         case -3:
68:             printf("Error: Invalid Message \n");
69:             return 0;
70:             break;
71:         case -9:
72:             printf("Error: Unable to Register \n");
73:             return 0;
74:             break;
75:         case -10:
76:             printf("Error: Invalid xTEDS \n");
77:             return 0;
78:             break;
79:         default:
80:             printf("Error: Unknown error code %d \n",result);

```

```

81:         return 0;
82:         break;
83:     }
84: }
85: sleep(5);
86: can.source = xTEDS.source;
87: result = 0;
88: result = can.Send();
89: if(result < 0)
90: {
91:     switch(result)
92:     {
93:         case -1:
94:             printf("Error: Send Error \n");
95:             return 0;
96:             break;
97:         case -2:
98:             printf("Error: Recv Error \n");
99:             return 0;
100:            break;
101:         case -3:
102:             printf("Error: Invalid Message \n");
103:             return 0;
104:             break;
105:         case -11:
106:             printf("Error: Unknown xTEDS \n");
107:             return 0;
108:             break;
109:         case -14:
110:             printf("Error: Invalid Cancel \n");
111:             return 0;
112:             break;
113:         default:
114:             printf("Error: Unknown error code %d \n",result);
115:             return 0;
116:             break;
117:     }
118: }
119: return 0;
120: }

```

File: sdm/app/test/DevicesTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5:
6: #include "../common/message/SDMReqReg.h"
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/message/SDMConsume.h"
9: #include "../common/message/SDMData.h"
10: #include "../common/MessageManager/MessageManager.h"
11:
12: const long my_port = 4123;
13:
14: int main (int argc, char ** argv)
15: {
16:     MessageManager mm;
17:     SDMInit(argc, argv);
18:     char buf[BUFSIZE];
19:
20:     mm.Async_Init(my_port);
21:
22:     SDMReqReg request;
23:     request.destination.setPort(my_port);
24:     request.reply = SDM_REQREG_CURRENT_AND_FUTURE;
25:     request.Send();
26:
27:     SDMRegInfo info;
28:     SDMConsume cons;
29:     SDMData data;
30:
31:     while (1)
32:     {
33:         if (mm.IsReady())
34:         {
35:             unsigned char type = mm.GetMessage(buf);
36:             switch (type)
37:             {
38:                 case SDM_RegInfo:
39:                 {
```

```

40:         if (info.Unmarshal(buf) > 0)
41:         {
42:             if (info.source.getPort() == PORT_PM || info.source.getPort() == PORT_TM ||
info.source.getPort() == PORT_DM)
43:                 continue;
44:                 cons.source = info.source;
45:                 cons.destination.setPort(my_port);
46:                 cons.msg_id = info.msg_id;
47:                 char provider[256];
48:                 char message[256];
49:                 cons.msg_id.IDToString(message, sizeof(message));
50:                 cons.source.IDToString(provider, sizeof(provider));
51:                 printf("Subscribing to %s from %s \n",message,provider);
52:                 cons.Send();
53:         }
54:     }
55:     break;
56:     case SDM_Data:
57:     {
58:         data.Unmarshal(buf);
59:         char src[128];
60:         char id[128];
61:         data.source.IDToString(src, sizeof(src));
62:         data.msg_id.IDToString(id, sizeof(id));
63:         printf("SDMData from %s id %s \n",src,id);
64:     }
65:     break;
66: }
67: }
68: else
69:     sleep(1);
70: }
71:
72: return 0;
73: }

```

File: sdm/app/test/TCPTester.cpp

```
1: #include <stdio.h>
2: #include <unistd.h>
3: #include <string.h>
4: #include "../common/message/SDMDeletesub.h"
5: #include "../common/message/SDMSubreqst.h"
6: #include "../common/message/SDMCommand.h"
7: #include "../common/message/SDMSerreqst.h"
8: #include "../common/message/SDMReady.h"
9: #include "../common/message/SDMConsume.h"
10: #include "../common/message/SDMCancel.h"
11: #include "../common/TCPcom.h"
12:
13:
14:
15: int main(int argc, char ** argv)
16: {
17:     int sock;
18:     int length;
19:     char msg_buf[BUFSIZE];
20:     SDMSubreqst sub_msg;
21:     SDMDeletesub delsub_msg;
22:     SDMCommand cmd_msg;
23:     SDMSerreqst ser_msg;
24:     SDMConsume cons_msg;
25:     SDMReady ready_msg;
26:     SDMCancel can_msg;
27:     char buf[11];
28:
29:     printf(" \n**Make sure the DM and SM have debug levels at least one to see the messages
received.** \n \n");
30:
31:     printf("Finding data manager...");
32:     sock = TCPconnect("127.0.0.1", PORT_DM);
33:     if (sock > 0)
34:         printf("Done. \n");
35:     else
36:     {
37:         printf("Error. Try again. \n");
38:         return -1;
```



```

39: }
40: //
41: // Send a consume message
42: cons_msg.msg_id.setInterface(5);
43: cons_msg.msg_id.setMessage(5);
44: length = cons_msg.Marshal(msg_buf);
45: printf("Sending SDMConsume message to DM...");
46: if (TCPsend (sock, msg_buf, length) > 0)
47:     printf("Done. \n");
48: else
49:     printf("Error. \n");
50:
51: //
52: // Send a ready message
53: sock = TCPconnect("127.0.0.1", PORT_DM);
54: length = ready_msg.Marshal(msg_buf);
55: printf("Sending SDMReady message to DM...");
56: if (TCPsend (sock, msg_buf, length) >=0)
57:     printf("Done. \n");
58: else
59:     printf("Error. \n");
60:
61: //
62: // Send a ready message in segments
63: sock = TCPconnect("127.0.0.1", PORT_DM);
64: printf("Sending SDMReady message to DM in segments...");
65:
66: if(TCPsend(sock, msg_buf, 8) > 0)
67:     printf("Sending header done. ...");
68: else
69:     printf("Error sending header. ...");
70:
71: sleep(1);
72: if(TCPsend(sock, msg_buf + 8, 13) >= 0)
73:     printf("Sending destination done. \n");
74: else
75:     printf("Error sending destination. \n");
76:
77: sleep(1);
78: if(TCPsend(sock, msg_buf + 21, 10) >= 0)
79:     printf("Sending destination done. \n");

```

```

80: else
81:     printf("Error sending destination. \n");
82:
83: //
84: // Send a command message
85: sock = TCPconnect("127.0.0.1", PORT_DM);
86: cmd_msg.command_id.setInterface(5);
87: cmd_msg.command_id.setMessage(5);
88: length = cmd_msg.Marshal(msg_buf);
89:
90: printf("Sending SDMCommand message...");
91: if (TCPSend (sock, msg_buf, length) >=0)
92:     printf("Done. \n");
93: else
94:     printf("Error. \n");
95:
96: sock = TCPconnect("127.0.0.1", PORT_DM);
97: can_msg.msg_id.setInterface(5);
98: can_msg.msg_id.setMessage(5);
99: length = can_msg.Marshal(msg_buf);
100:
101: printf("Sending SDMSerreqst message...");
102: if (TCPSend (sock, msg_buf, length) >=0)
103:     printf("Done. \n");
104: else
105:     printf("Error. \n");
106:
107: fflush(NULL);
108:
109: //TEST SENSOR MANAGER's TCP LISTENER
110:
111: printf("Finding sensor manager...");
112: sock = TCPconnect("127.0.0.1", PORT_SM);
113: if (sock >= 0)
114:     printf("Done. \n");
115: else
116: {
117:     printf("Error. Try again. \n");
118:     return 1;
119: }
120:

```

```

121:  sub_msg.msg_id.setInterface(5);
122:  sub_msg.msg_id.setMessage(5);
123:  sub_msg.source.setSensorID(1);
124:  length = sub_msg.Marshal(msg_buf);
125:
126:  printf("Sending SDMSubreqst message to SM...");
127:  if (TCPsend (sock, msg_buf, length) >=0)
128:      printf("Done. \n");
129:  else
130:      printf("Error. \n");
131:
132:  sock = TCPconnect("127.0.0.1", PORT_SM);
133:  delsub_msg.msg_id.setInterface(5);
134:  delsub_msg.msg_id.setMessage(5);
135:  length = delsub_msg.Marshal(msg_buf);
136:
137:  printf("Sending SDMDeletesub message to SM...");
138:  if (TCPsend (sock, msg_buf, length) >=0)
139:      printf("Done. \n");
140:  else
141:      printf("Error. \n");
142:
143:  sock = TCPconnect("127.0.0.1", PORT_SM);
144:  cmd_msg.command_id.setInterface(5);
145:  cmd_msg.command_id.setMessage(5);
146:  length = cmd_msg.Marshal(msg_buf);
147:
148:  printf("Sending SDMCommand message to SM...");
149:  if (TCPsend (sock, msg_buf, length) >=0)
150:      printf("Done. \n");
151:  else
152:      printf("Error. \n");
153:
154:  sock = TCPconnect("127.0.0.1", PORT_SM);
155:  ser_msg.command_id.setInterface(5);
156:  ser_msg.command_id.setMessage(5);
157:  length = ser_msg.Marshal(msg_buf);
158:
159:  printf("Sending SDMSerreqst message to SM...");
160:  if (TCPsend (sock, msg_buf, length) >=0)
161:      printf("Done. \n");

```

```
162:  else
163:      printf("Error. \n");
164:
165:  fflush(NULL);
166:  return 0;
167: }
```

File: sdm/app/test/ModificationSearchSubTest.cpp

```
1: #include "../common/message/SDMSearch.h"
2: #include "../common/message/SDMSearchReply.h"
3: #include "../common/MessageManager/MessageManager.h"
4: #include "../common/message/SDMData.h"
5: #include "../common/message/SDMConsume.h"
6: #include <string.h>
7: #include <sys/types.h>
8: #include <sys/stat.h>
9: #include <fcntl.h>
10: #include <unistd.h>
11: #include <stdio.h>
12: #include <stdlib.h>
13: #include <pthread.h>
14: #include <signal.h>
15:
16: #define ORIENTATION_SEARCH_ID 1
17:
18: long my_port = 4020;
19: int sigint_count = 0;
20:
21: void SigHandler(int signum);
22: int main(int argc, char** argv)
23: {
24:     SDMSearch s;
25:     SDMSearchReply reply;
26:     SDMData dat;
27:     SDMConsume con;
28:     int result = 0;
29:     char buf[BUFSIZE];
30:     long length;
31:     MessageManager mm;
32:     int cur = 0;
33:     int count = 0;
34:     char c;
35:     unsigned long SID;
36:
37:     signal(SIGINT, SigHandler);
38:     SDMInit(argc, argv);
39:     my_port = getPort();
```

```

40: if(my_port == SDM_PM_NOT_AVAILABLE)
41: {
42:     printf("Unable to get a port from the Process Manager! \n");
43:     exit(0);
44: }
45: mm.Async_Init(my_port);
46: sleep(1);
47: s.destination.setPort(my_port);
48: s.destination.setAddress(DataManager.getAddress()); //This application must be run on the same
node as the DM
49: s.reply = SDM_SEARCH_CURRENT_AND_FUTURE;
50:
51: printf(" \nSending SDMSearch for Device orientation \n");
52: memset(s.reg_expr,0,512);
53: strcpy(s.reg_expr,".*?<Orientation.*?axis= \"(.*)\" \".*\");
54: s.id = ORIENTATION_SEARCH_ID;
55: s.Send();
56:
57: con.destination = s.destination;
58: con.source.setPort(PORT_DM);
59: con.source.setAddress(DataManager.getAddress());
60: con.source.setSensorID(1);
61: con.msg_id = 0x103;
62: con.Send();
63:
64: con.msg_id = 0x104;
65: con.Send();
66: while(1)
67: {
68:     if (mm.IsReady())
69:     {
70:         c = mm.GetMessage(buf,length);
71:         cur = 0;
72:         switch(c)
73:         {
74:             case SDM_SearchReply:
75:                 result = reply.Unmarshal(buf);
76:                 if(result == SDM_NO_FURTHER_DATA_PROVIDER)
77:                     continue;
78:                 printf(" \n");
79:                 if (reply.id == ORIENTATION_SEARCH_ID)

```

```

80:             printf("Reply for orientation search: \n");
81:         if(count != 0)
82:             printf(" \n");
83:             printf("Match(es)                from                %lu:%lu:%d
\n",reply.source.getSensorID(),reply.source.getAddress(),reply.source.getPort());
84:             count++;
85:             while(reply.captured_matches[cur]!=0)
86:             {
87:                 if(strlen(reply.captured_matches+cur)<32)
88:                     printf("Match is %s \n",reply.captured_matches+cur);
89:                 else
90:                     printf("Match length of %d exceeds name length restriction of 32
\n",strlen(reply.captured_matches+cur));
91:                 cur += strlen(reply.captured_matches+cur)+1;
92:             }
93:             break;
94:         case SDM_Data:
95:             result = dat.Unmarshal(buf);
96:             if(dat.length > 4)
97:             {
98:                 memcpy(&SID,&dat.msg[1],4);
99:                 printf("xTEDS change of type %d on SID %ld \n",dat.msg[0],SID);
100:             }
101:             else
102:             {
103:                 memcpy(&SID,&dat.msg[0],4);
104:                 printf("xTEDS modificaton of SID %ld \n",SID);
105:             }
106:             break;
107:         default:
108:             printf("Unexpected message of type %c \n",c);
109:     }
110: }
111: else
112: {
113:     printf(".");
114:     sleep(1);
115:     fflush(NULL);
116: }
117: }
118: }

```

```
119: void SigHandler(int signum)
120: {
121:     SDMSearch search;
122:     printf("Sending search subscription cancellation for interface searches only. \n");
123:     search.reply = SDM_SEARCH_CANCEL;
124:     search.destination.setPort(my_port);
125:     strcpy(search.reg_expr, ".*?<Orientation.*?axis= \"(.*)\" .*");
126:     search.Send();
127:     sigint_count++;
128:     exit(0);
129: }
130:
```


File: sdm/app/test/KillTester.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <stdlib.h>
4: #include <unistd.h>
5: #include "../common/message/SDMKill.h"
6: #include "../common/message/SDMReqReg.h"
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/message/SDMCommand.h"
9: #include "../common/message/SDMService.h"
10: #include "../common/message/SDMConsume.h"
11: #include "../common/message/SDMData.h"
12: #include "../common/MessageManipulator/MessageManipulator.h"
13: #include "../common/MessageManager/MessageManager.h"
14:
15: #define TM_NAME_TO_PID      1
16: #define TM_TASK_STARTED_MESSAGE 2
17:
18: long myPort;
19:
20: int main (int argc, char** argv)
21: {
22:     SDMInit(argc, argv);
23:     MessageManipulator TMNameToPID;
24:     MessageManipulator TMTaskStartedMessage;
25:
26:     SDMComponent_ID TaskManagerID;
27:
28:     MessageManager mm;
29:     myPort = getPort();
30:     if (myPort == SDM_PM_NOT_AVAILABLE)
31:     {
32:         printf("PM not available. \n");
33:         return -1;
34:     }
35:     mm.Async_Init(myPort);
36:
37:     const char* TaskName = "producer";
38:
39:     // Send the ReqReg for TM's name to SDM pid service
```

```

40: SDMReqReg ReqReg;
41: strcpy(ReqReg.item_name, "NameToPID");
42: ReqReg.reply = SDM_REQREG_CURRENT;
43: ReqReg.destination.setPort(myPort);
44: ReqReg.id = TM_NAME_TO_PID;
45: ReqReg.Send();
46:
47: // Send the ReqReg for TM's task started notification
48: strcpy(ReqReg.item_name, "TaskStarted");
49: ReqReg.id = TM_TASK_STARTED_MESSAGE;
50: ReqReg.Send();
51:
52: char MessageBuffer[BUFSIZE];
53: while (1)
54: {
55:     char MessageType = mm.BlockGetMessage(MessageBuffer);
56:     switch(MessageType)
57:     {
58:         case SDM_RegInfo:
59:         {
60:             SDMRegInfo InfoReply;
61:             if (InfoReply.Unmarshal(MessageBuffer) ==
SDM_NO_FURTHER_DATA_PROVIDER)
62:                 break;
63:             if (InfoReply.id == TM_TASK_STARTED_MESSAGE)
64:             {
65:                 // Set the Message Manipulator
66:                 TMTaskStartedMessage.setMsgDef(InfoReply);
67:                 TaskManagerID = InfoReply.source;
68:
69:                 // Consume the task started notification
70:                 SDMConsume Consume;
71:                 Consume.source = InfoReply.source;
72:                 Consume.msg_id = InfoReply.msg_id;
73:                 Consume.destination.setPort(myPort);
74:                 Consume.Send();
75:             }
76:             else if (InfoReply.id == TM_NAME_TO_PID)
77:             {
78:                 // Set the Message Manipulator
79:                 TMNameToPID.setMsgDef(InfoReply);

```

```

80:         TaskManagerID = InfoReply.source;
81:     }
82: }
83: break;
84: case SDM_Data:
85: {
86:     SDMDData Data;
87:     Data.Unmarshal(MessageBuffer);
88:     if (Data.msg_id == TMTaskStartedMessage.getMsgID(DATAMSG))
89:     {
90:         int Length;
91:         char* MessageTaskName = (char*)TMTaskStartedMessage.getArray("TaskName",
Data, DATAMSG, Length);
92:         if (MessageTaskName == NULL)
93:             break;
94:         // See if the task we want to kill was just queued
95:         if (strcmp(MessageTaskName, TaskName)==0)
96:         {
97:             // Request the task's SDM pid
98:             SDMSERVICE Service;
99:             Service.source = TaskManagerID;
100:             Service.destination.setPort(myPort);
101:             Service.command_id = TMNameToPID.getMsgID(COMMANDMSG);
102:             TMNameToPID.setArray("TaskName", Service, TaskName,
strlen(TaskName)+1);
103:             Service.length = TMNameToPID.getLength(COMMANDMSG);
104:             Service.Send();
105:         }
106:     }
107:     else if (Data.msg_id == TMNameToPID.getMsgID(DATAMSG))
108:     {
109:         // Kill the received PID
110:         unsigned int PID = TMNameToPID.getUINT32Value("SDMTaskPID", Data,
DATAMSG);
111:         printf("Killing pid %u \n",PID);
112:         SDMKill Kill;
113:         Kill.PID = PID;
114:         sleep(1);
115:         Kill.Send();
116:     }
117: }
118: break;

```

```
119:     }  
120:  
121: }  
122: return 0;  
123: }
```

File: sdm/app/test/FileServiceTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <unistd.h>
4:
5: #include "../common/message/SDMCommand.h"
6: #include "../common/message/SDMReqReg.h"
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/message/SDMData.h"
9: #include "../common/message/SDMService.h"
10: #include "../common/MessageManager/MessageManager.h"
11:
12: const int FILEPATH_SIZE = 512;
13: //Status code definitions
14: const unsigned char SUC_OPERATION_OK = 1;
15: const unsigned char FLT_INVALID_HANDLE = 2;
16: const unsigned char FLT_FILE_NOT_AVAILABLE = 3;
17: const unsigned char FLT_INVALID_OFFSET = 4;
18: const unsigned char FLT_COULD_NOT_OBTAIN_HANDLE = 5;
19: const unsigned char FLT_WRITE_FAILURE = 6;
20: const unsigned char FLT_INVALID_WRITE_MODE = 7;
21: //File mode flags
22: const unsigned char READ_ONLY = 1;
23: const unsigned char WRITE_ONLY_OFFSET = 2;
24: const unsigned char WRITE_ONLY_APPEND = 3;
25: const unsigned char READ_WRITE_OFFSET = 4;
26: const unsigned char READ_WRITE_APPEND = 5;
27: //Message IDs as defined in the xTEDS
28: const SDMMMessage_ID SerReadPortion(1,6); //Identifier for reading a portion of the file
29: const SDMMMessage_ID SerWritePortion(1,9); //Write portion request id
30: const SDMMMessage_ID SerOpenFileHandle(1,1); //Identifier for open file handle service request
    message
31: const SDMMMessage_ID RplyOpenFileHandle(1,2); //Identifier for open file handle response
    message
32: const SDMMMessage_ID RplyReadReply(1,7); //Read portion reply id
33: const SDMMMessage_ID RplyWriteReply(1,10); //Write reply message
34: const SDMMMessage_ID FltHandleOpenFailed(1,3); //Identifier for open file handle fault
35: const SDMMMessage_ID FltHandleCloseFailed(1,4); //Identifier for close handle fault
36: const SDMMMessage_ID FltReadPortionError(1,8); //Read portion fault id
37: const SDMMMessage_ID FltWritePortionError(1,11); //Write portion error id
```

```

38: const SDMMMessage_ID CmdCloseFileHandle(1,4); //Identifier for close file handle command
message
39:
40: SDMComponent_ID FileServiceID;
41: MessageManager mm;
42:
43: void GetFSID();
44: long GetFileHandle(const char *filename, unsigned char flags);
45: void CloseFileHandle(long handle);
46: bool ReadFile(long handle, const char *filename, unsigned int offset, unsigned int length);
47: bool WriteFile(long handle, const char *filename, unsigned int offset, unsigned int length, const char
*buffer, unsigned char mode = 0);
48:
49: int main(int argc, char ** argv)
50: {
51: if (argc < 4)
52: {
53:     printf("Usage: \n \t%s [DM-Address] [TM-Address] [PID] \n",argv[0]);
54:     return 0;
55: }
56: SDMInit(argc, argv);
57:
58: mm.Async_Init(4050);
59: GetFSID();
60: const char file1[] = "tm.cpp";
61: const char file2[] = "DM.cpp";
62: const char file3[] = "sm.cpp";
63: const char file4[] = "pm.cpp";
64: const char file5[] = "test_file.txt";
65: const char file6[] = "test/test_file2.txt";
66: const char file7[] = "test/tm.cpp";
67:
68: long Handle1 = GetFileHandle(file1, READ_WRITE_OFFSET);
69: usleep(200);
70: long Handle2 = GetFileHandle(file2, READ_ONLY);
71: usleep(200);
72: long Handle3 = GetFileHandle(file3, READ_ONLY);
73: usleep(200);
74: long Handle4 = GetFileHandle(file4, READ_ONLY);
75: usleep(200);
76: long Handle5 = GetFileHandle(file5, WRITE_ONLY_OFFSET);

```

```

77: usleep(200);
78: long Handle6 = GetFileHandle(file6, READ_ONLY);
79: long temp;
80:
81: if (Handle1 <= 0 || Handle2 <=0 || Handle3 <=0 || Handle4 <=0 || Handle5 <=0 || Handle6 <=0)
82: {
83:     printf("Handle <= 0 \n");
84:     return 0;
85: }
86: printf("-----TESTING FILE HANDLE CASES----- \n");
87: printf("Attempting to open %s for reading...",file1);
88: temp = GetFileHandle(file1, READ_ONLY);
89: if (temp < 0)    printf("failed as expected. \n");
90: else{ printf(" \nERROR: test failed. \n"); return 0; }
91:
92: printf("Attempting to open %s for writing...",file2);
93: temp = GetFileHandle(file2, WRITE_ONLY_APPEND);
94: if (temp < 0)    printf("failed as expected. \n");
95: else{ printf(" \nERROR: test failed. \n"); return 0; }
96:
97: printf("Attempting to open %s for reading...",file5);
98: temp = GetFileHandle(file5, READ_ONLY);
99: if (temp < 0)    printf("failed as expected. \n");
100: else{ printf(" \nERROR: test failed. \n"); return 0; }
101:
102: printf("Attempting to open %s for reading and writing...",file4);
103: temp = GetFileHandle(file4, READ_WRITE_APPEND);
104: if (temp < 0)    printf("failed as expected. \n");
105: else{ printf(" \nERROR: test failed. \n"); return 0; }
106:
107: printf("Attempting to open %s for reading... \n",file2);
108: temp = GetFileHandle(file2, READ_ONLY);
109: if (temp < 0) { printf("Test failed. \n"); return 0; }
110: else { CloseFileHandle(temp); printf(" \tTest Passed. \n"); }
111: printf("-----DONE TESTING FILE HANDLE CASES----- \n");
112:
113: printf("-----TESTING READ OPERATIONS----- \n");
114: int offset = 3;
115: int length = 10;
116: printf("Testing read from %s (offset %d, length %d) \n", file1, offset, length);
117: if (ReadFile(Handle1, "", offset, length))

```

```

118:     printf(" \tTest Passed. \n");
119: else
120: { printf(" \tTest Failed. \n"); return 0; }
121:
122: offset = 300;
123: length = 20;
124: printf("Testing read from %s (offset %d, length %d) \n", file2, offset, length);
125: if (ReadFile(Handle2, "", offset, length))
126:     printf(" \tTest Passed. \n");
127: else
128: { printf(" \tTest Failed. \n"); return 0; }
129:
130: offset = 3;
131: length = 10;
132: printf("Testing read from %s (offset %d, length %d) using filename (shouldn't work) \n", file1,
offset, length);
133: if (!ReadFile(0, file1, offset, length))
134:     printf(" \tTest Passed. \n");
135: else
136: { printf(" \tTest Failed. \n"); return 0; }
137:
138: offset = 3;
139: length = 10;
140: printf("Testing read from %s (offset %d, length %d) (shouldn't work) \n", file5, offset, length);
141: if (!ReadFile(0, file5, offset, length))
142:     printf(" \tTest Passed. \n");
143: else
144: { printf(" \tTest Failed. \n"); return 0; }
145:
146: offset = 3;
147: length = 10;
148: printf("Testing read from %s (offset %d, length %d) \n", file6, offset, length);
149: if (ReadFile(Handle6, "", offset, length))
150:     printf(" \tTest Passed. \n");
151: else
152: { printf(" \tTest Failed. \n"); return 0; }
153: printf("-----DONE TESTING READ OPERATIONS----- \n");
154:
155: printf("-----TESTING WRITE OPERATIONS----- \n");
156: offset = 100;
157: length = 20;

```



```

158: printf("Testing read/write for %s (offset %d, length %d) \n", file1, offset, length);
159: if (WriteFile(Handle1, "", offset, length, "THIS IS A TEST WRITE"))
160: {
161:     if (ReadFile(Handle1, "", offset, length))
162:         printf(" \tTest passed. \n");
163:     else
164:         { printf(" \tTest failed. \n"); return 0; }
165: }
166: else
167: { printf(" \tTest failed. \n"); return 0; }
168:
169: offset = 10;
170: length = 20;
171: printf("Testing write append for %s (offset %d, length %d) \n", file5, offset, length);
172: if (WriteFile(Handle5, "", offset, length, "THIS IS A TEST WRITE"))
173:     printf(" \tTest passed. \n");
174: else
175: { printf(" \tTest failed. \n"); return 0; }
176:
177: offset = 0;
178: length = 20;
179: printf("Testing write append for %s (offset %d, length %d) \n", file5, offset, length);
180: if (WriteFile(Handle5, "", offset, length, "THIS IS A TEST WRITE"))
181:     printf(" \tTest passed. \n");
182: else
183: { printf(" \tTest failed. \n"); return 0; }
184:
185: offset = 100;
186: length = 20;
187: printf("Testing write by filename for %s (offset %d, length %d) (won't write) \n", file5, offset,
length);
188: if (!WriteFile(0, file5, offset, length, "THIS IS A TEST WRITE", WRITE_ONLY_APPEND))
189:     printf(" \tTest passed. \n");
190: else
191: { printf(" \tTest failed. \n"); return 0; }
192:
193: offset = 100;
194: length = 20;
195: printf("Testing write by filename for %s (offset %d, length %d) \n", file7, offset, length);
196: if (WriteFile(0, file7, offset, length, "THIS IS A TEST WRITE", WRITE_ONLY_APPEND))
197:     printf(" \tTest passed. \n");

```

```

198:     else
199:     {   printf("\tTest failed. \n"); return 0; }
200:     printf("-----DONE TESTING WRITE OPERATIONS----- \n");
201:
202:     CloseFileHandle(Handle1);
203:     CloseFileHandle(Handle2);
204:     CloseFileHandle(Handle3);
205:     CloseFileHandle(Handle4);
206:     CloseFileHandle(Handle5);
207:     CloseFileHandle(Handle6);
208:
209:     return 0;
210: }
211: void GetFSID()
212: {
213:     printf("Requesting FileServiceID...");
214:     SDMReqReg request;
215:     strcpy(request.interface, "FileServiceInterface");
216:     strcpy(request.item_name, "OpenFileHandle");
217:     request.destination.setPort(4050);
218:     request.Send();
219:
220:     SDMRegInfo info;
221:     char buffer[BUFSIZE];
222:     bool found = false;
223:     while (!found)
224:     {
225:         if (mm.IsReady())
226:         {
227:             char type = mm.GetMessage(buffer);
228:             if (type == SDM_RegInfo)
229:             {
230:                 if (info.Unmarshal(buffer) > 0)
231:                 {
232:                     found = true;
233:                 }
234:             }
235:         }
236:         else
237:             sleep(1);
238:     }

```

```

239:  FileServiceID = info.source;
240:  char buf[256];
241:  FileServiceID.IDToString(buf, sizeof(buf));
242:  printf("Found (%s). \n",buf);
243: }
244: long GetFileHandle(const char *filename, unsigned char flags)
245: {
246:     printf("Requesting file handle..."); fflush(NULL);
247:     //
248:     //Obtain a file handle
249:     SDMService request;
250:     request.command_id = SerOpenFileHandle;
251:     request.source = FileServiceID;
252:     request.destination.setPort(4050);
253:     strcpy(request.data, filename);
254:     PUT_UCHAR(&request.data[FILEPATH_SIZE], flags);
255:     request.length = FILEPATH_SIZE+1;
256:     request.Send();
257:
258:     SDMData data;
259:     char buffer[BUFSIZE];
260:     bool found = false;
261:     while (!found)
262:     {
263:         if (mm.IsReady())
264:         {
265:             char type = mm.GetMessage(buffer);
266:             if (type == SDM_Data)
267:             {
268:                 if (data.Unmarshal(buffer) > 0)
269:                 {
270:                     found = true;
271:                 }
272:             }
273:         }
274:         else
275:             sleep(1);
276:     }
277:     unsigned short handle;
278:     if (data.msg_id == RplyOpenFileHandle)
279:     {

```

```

280:     handle = GET_USHORT(data.msg+FILEPATH_SIZE);
281:     printf("Handle received for %s (%hu) \n",data.msg,handle);
282:     return handle;
283: }
284: else if (data.msg_id == FltHandleOpenFailed)
285: {
286:     unsigned char err_code = GET_UCHAR(data.msg+FILEPATH_SIZE);
287:     printf("Error receiving handle (%c) \n",err_code);
288: }
289: return -1;
290: }
291: void CloseFileHandle(long handle)
292: {
293:     SDMCommand cmd;
294:     unsigned short Handle_m = (unsigned short) handle;
295:     printf("Closing file handle %hu \n",Handle_m);
296:     cmd.source = FileServiceID;
297:     cmd.destination.setPort(4050);
298:     cmd.command_id = CmdCloseFileHandle;
299:     PUT_USHORT(cmd.data, Handle_m);
300:     cmd.length = 2;
301:     cmd.Send();
302: }
303: bool ReadFile(long handle, const char *filename, unsigned int offset, unsigned int length)
304: {
305:     printf("Reading file... \n");
306:     SDMService request;
307:     request.command_id = SerReadPortion;
308:     request.source = FileServiceID;
309:     request.destination.setPort(4050);
310:
311:     PUT_USHORT(request.data, handle);
312:     strcpy(request.data+2, filename);
313:     PUT_UINT(request.data+2+FILEPATH_SIZE, offset);
314:     PUT_UINT(request.data+2+FILEPATH_SIZE+4, length);
315:     request.length = 2+FILEPATH_SIZE+4+4;
316:     request.Send();
317:
318:     SDMData data;
319:     char buffer[BUFSIZE];
320:     bool done = false;

```

```

321: unsigned short num_segments = 0;
322: unsigned short cur_segment = 0;
323: unsigned int num_received = 0;
324: char buf[BUFSIZE];
325: unsigned int cur_length = 0;
326: while (!done)
327: {
328:     if (mm.IsReady())
329:     {
330:         char type = mm.GetMessage(buffer);
331:         if (type == SDM_Data)
332:         {
333:             if (data.Unmarshal(buffer) > 0)
334:             {
335:                 if (data.msg_id == RplyReadReply)
336:                 {
337:                     if (num_segments == 0)
338:                     {
339:                         num_segments = GET_USHORT(data.msg+2+2+FILEPATH_SIZE);
340:                         printf("Number of segments is %hu. \n",num_segments);
341:                     }
342:                     num_received++;
343:                     cur_length = GET_UINT(data.msg+FILEPATH_SIZE+6);
344:                     memset(buf, 0, sizeof(buf));
345:                     strncpy(buf, data.msg+FILEPATH_SIZE+10, cur_length);
346:                     cur_segment = GET_USHORT(data.msg+FILEPATH_SIZE+2);
347:                     printf("Received segment %hu length is %u \n",cur_segment,cur_length);
348:                     printf("Contents of segment %hu: %s \n",cur_segment, buf);
349:
350:                     if (num_received == num_segments)
351:                         done = true;
352:                 }
353:                 else
354:                     return false;
355:             }
356:         }
357:     }
358:     else
359:         sleep(1);
360: }
361: return true;

```

```

362: }
363: bool WriteFile(long handle, const char *filename, unsigned int offset, unsigned int length, const char
*write_buffer, unsigned char mode)
364: {
365:     printf("Writing file... \n");
366:     SDMService request;
367:     request.command_id = SerWritePortion;
368:     request.source = FileServiceID;
369:     request.destination.setPort(4050);
370:
371:     PUT_USHORT(request.data, handle);
372:     strcpy(request.data+2, filename);
373:     PUT_UCHAR(&request.data[2+FILEPATH_SIZE], mode);
374:     PUT_UINT(request.data+2+FILEPATH_SIZE+1, offset);
375:     PUT_UINT(request.data+2+FILEPATH_SIZE+5, length);
376:     strcpy(request.data+11+FILEPATH_SIZE, write_buffer);
377:     request.length = 11+FILEPATH_SIZE+strlen(write_buffer);
378:     request.Send();
379:
380:     SDMData data;
381:     bool done = false;
382:     char buffer[BUFSIZE];
383:     char file[FILEPATH_SIZE];
384:     while (!done)
385:     {
386:         if (mm.IsReady())
387:         {
388:             char type = mm.GetMessage(buffer);
389:             if (type == SDM_Data)
390:             {
391:                 if (data.Unmarshal(buffer) > 0)
392:                 {
393:                     if (data.msg_id == RplyWriteReply)
394:                     {
395:                         unsigned short handle = GET_USHORT(data.msg);
396:                         strcpy(file, data.msg+2);
397:                         unsigned char status = GET_UCHAR(data.msg+FILEPATH_SIZE+2);
398:                         printf(" Success for handle %hu, file %s, status %hhu. \n",handle, file,
status);
399:                         done = true;
400:                     }

```

```

401:         else if (data.msg_id == FltWritePortionError)
402:         {
403:             unsigned short handle = GET_USHORT(data.msg);
404:             strcpy(file, data.msg+2);
405:             unsigned char status = GET_UCHAR(data.msg+FILEPATH_SIZE+2);
406:             printf(" Error for handle %hu, file %s, status %hhu. \n", handle, file, status);
407:             return false;
408:         }
409:     }
410: }
411: }
412: else
413:     sleep(1);
414: }
415: return true;
416: }

```

File: sdm/app/test/TestRegEx.cpp

```
1: //To be used in conjunction with the examples Converter and Producer
2:
3: #include "../common/message/SDMData.h"
4: #include "../common/message/SDMService.h"
5: #include "../common/message/SDMConsume.h"
6: #include "../common/message/SDMRegInfo.h"
7: #include "../common/message/SDMReqReg.h"
8: #include "../common/MessageManipulator/MessageManipulator.h"
9: #include "../common/MessageManager/MessageManager.h"
10:
11: #include <string.h>
12: #include <unistd.h>
13: #include <stdio.h>
14: #include <sys/types.h>
15: #include <sys/wait.h>
16: #include <signal.h>
17:
18: #define DATA_PROVIDER    1
19: #define SERVICE_PROVIDER 2
20:
21: long my_port;
22: SDMComponent_ID data_provider;
23: SDMComponent_ID service_provider;
24: SDMMessage_ID data_msg(0,0);
25: SDMMessage_ID service_msg(0,0);
26: bool d_sent = false;
27: bool s_sent = false;
28:
29: MessageManipulator data_manipulator;
30: MessageManipulator service_manipulator;
31:
32: void DataHandler(SDMData& dat,long length)
33: {
34:     SDMService request;
35:     short int_value;
36:     float float_value;
37:     static float geo_average = 0;
38:
39:     if((dat.source == data_provider)&&(dat.msg_id == data_msg))
```



```

40: {
41:     //marshal appropriate service message
42:     //copy sensor id into service request
43:     request.source=service_provider;
44:     //copy msg id into service request
45:     request.command_id=service_msg;
46:     //Set service request receive
47:     request.destination.setPort(my_port);
48:     //set the length we are sending
49:     request.length = sizeof(short);
50:     //copy integer data into service request
51:     if(service_provider.getSensorID()==0)
52:     {
53:         printf("No converter is available \n");
54:         return;
55:     }
56:     int_value = data_manipulator.getUINT16Value("data",dat,DATAMSG);
57:     service_manipulator.setValue("converttee",request,int_value);
58:     //send request
59:     request.Send();
60: }
61: else if((dat.source == service_provider))
62: {
63:     float_value = service_manipulator.getFloat32Value("data",dat,DATAMSG);
64:     geo_average += float_value;
65:     float_value /=2;
66:     //extract and display results
67:     printf("Running Average -- %f \n",geo_average);
68: }
69: }
70:
71: void RegInfoHandler(SDMRegInfo& info)
72: {
73:     SDMConsume consume;
74:     SDMReqReg req_reg;
75:
76:     //Set the port we will be receiving on
77:     consume.destination.setPort(my_port);
78:     //copy the sensor id into the consume message
79:     consume.source = info.source;
80:     //copy the msg id into the consume message

```

```

81: consume.msg_id=info.msg_id;
82:
83: switch(info.id)
84: {
85: case DATA_PROVIDER:
86:     if(info.type == 1)
87:     {
88:         data_provider.setSensorID(0);
89:         printf("Data provider failed . . . ");
90:         printf("Searching for new provider \n");
91:         //request info on integer data providers,
92:         //the DM will repost software tasks that
93:         // could satisfy our requirements
94:
95:         //Set variable name
96:         strcpy(req_reg.item_name,"d.*");
97:         //Set the quallist can be empty
98:         strcpy(req_reg.quallist,"< FORMAT= \"UI \"/>");
99:         req_reg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
100:         req_reg.destination.setPort(my_port);
101:         req_reg.id = DATA_PROVIDER;
102:         req_reg.Send();
103:     }
104:     else
105:     {
106:         if(data_provider.getSensorID() == 0)
107:         {
108:             data_provider = info.source;
109:             data_msg = info.msg_id;
110:             data_manipulator.setMsgDef(info.msg_def);
111:             printf("New Data provider found");
112:             printf(" \n \t%s \n",info.msg_def);
113:             //Send the consume message
114:             consume.Send();
115:         }
116:         else
117:         {
118:             printf("Data provider found");
119:             printf(" - not used \n \t%s \n",info.msg_def);
120:         }
121:     }

```

```

122:     break;
123: case SERVICE_PROVIDER:
124:     if(info.type == SDM_REGINFO_CANCELLATION)
125:     {
126:         service_provider.setSensorID(0);
127:         printf("Service provider failed . . . ");
128:         printf("Searching for new provider \n");
129:         //request info on service providers,
130:         //the DM will repost software tasks
131:         //that could satisfy our requirements
132:
133:         //Set var name to convert which we already know
134:         strcpy(req_reg.item_name,"converter");
135:         //We will not be using wildcards
136:         req_reg.qualist[0] = '\0';
137:         req_reg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
138:         req_reg.destination.setPort(my_port);
139:         req_reg.id = SERVICE_PROVIDER;
140:         req_reg.Send();
141:     }
142:     else
143:     {
144:         if(service_provider.getSensorID() == 0)
145:         {
146:             service_provider = info.source;
147:             service_msg = info.msg_id;
148:             service_manipulator.setMsgDef(info.msg_def);
149:             printf("New Service provider found");
150:             printf("\n \t%s \n",info.msg_def);
151:         }
152:         else
153:         {
154:             printf("Service provider found - ");
155:             printf("not used \n \t%s \n",info.msg_def);
156:         }
157:     }
158:     break;
159: }
160:
161: }
162:

```

```

163: int main(int argc,char** argv)
164: {
165:     MessageManager mm;
166:     SDMDData dat;
167:     SDMRegInfo info;
168:     SDMReqReg req_reg;
169:     char buf[BUFSIZE];
170:     long length;
171:
172:     //initialize consumer
173:     SDMInit(argc,argv);
174:     my_port = getPort();
175:     mm.Async_Init(my_port);
176:
177:     printf("Consumer listening on port %ld \n",my_port);
178:
179:     while(1)
180:     {
181:         if (mm.IsReady())
182:         {
183:             switch(mm.GetMessage(buf,length))
184:             {
185:                 case SDM_Data:
186:                     dat.Unmarshal(buf,length);
187:                     DataHandler(dat,length);
188:                     break;
189:                 case SDM_RegInfo:
190:                     if(length > 1)
191:                     {
192:                         info.Unmarshal(buf);
193:                         RegInfoHandler(info);
194:                     }
195:                     break;
196:                 default:
197:                     printf("Unexpected message \n");
198:             }
199:         }
200:         else
201:         { //check for data and service providers
202:             if(data_provider.getSensorID() == 0 && d_sent == false)
203:             {

```

```

204:          //request info on integer data providers
205:          //Set variable name
206:          strcpy(req_reg.item_name,"d.*");
207:          //Set the quallist can be empty
208:          strcpy(req_reg.quallist,"< FORMAT= \"UI \"/>");
209:          req_reg.reply
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
210:          req_reg.destination.setPort(my_port);
211:          req_reg.id = DATA_PROVIDER;
212:          req_reg.Send();
213:          printf("Searching for new data provider \n");
214:          d_sent = true;
215:          sleep(2);
216:      }
217:      if(service_provider.getSensorID() == 0 && s_sent == false)
218:      {
219:          //request info on service providers
220:          //Set var name to convert which we already know
221:          strcpy(req_reg.item_name,"converter*");
222:          //We will not be using wildcards
223:          req_reg.quallist[0] = '\0';
224:          req_reg.reply
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
225:          req_reg.destination.setPort(my_port);
226:          req_reg.id = SERVICE_PROVIDER;
227:          req_reg.Send();
228:          printf("Searching for new service provider \n");
229:          s_sent = true;
230:          sleep(2);
231:      }
232:      usleep(1000);
233:  }
234:  }
235: }

```

File: sdm/app/test/MinimalxTEDSgetIP.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMData.h"
3: #include "../common/message/SDMCancelxTEDS.h"
4: #include "../common/message/SDMService.h"
5: #include "../common/MessageManager/MessageManager.h"
6: #include <string.h>
7: #include <sys/types.h>
8: #include <sys/stat.h>
9: #include <fcntl.h>
10: #include <unistd.h>
11: #include <stdio.h>
12: #include <stdlib.h>
13: #include <pthread.h>
14: #include <arpa/inet.h>
15:
16: void RegisterxTEDS();
17: void CancelxTEDS();
18: void GetComponentID();
19:
20: long my_port;
21: long my_sensorID;
22: long my_PID;
23: MessageManager mm;
24:
25: int main(int argc, char** argv)
26: {
27:     SDMData dat;
28:
29:
30:     SDMInit(argc, argv);
31:     my_port = getPort();
32:     if (my_port < 0)
33:     {
34:         printf("Error requesting port. \n");
35:         return -1;
36:     }
37:     mm.Async_Init(my_port);
38:     RegisterxTEDS();
39:     GetComponentID();
```

```

40: CancelxTEDS();
41: }
42:
43: void RegisterxTEDS()
44: {
45:   SDMxTEDS  xteds; // create an xTEDS registration message
46:
47:   strcpy (xteds.xTEDS,"<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS name=
  \"Minimal_xTEDS \" version= \"2.0 \"> \n \n \t<Application name= \"MinimalxTEDS \" kind=
  \"Software \"/> \n \n \t<Interface name= \"Min_Interface \" id= \"1 \"/> \n \n</xTEDS> \n"); //      set
  xTEDS
48:
49:   xteds.source.setSensorID(1);          // set the id of this application
50:   xteds.source.setPort(my_port);
51:   printf("Registering producer xTEDS on port %ld \n",my_port);
52:   xteds.Send();                        // register with the SDM
53: }
54:
55: void GetComponentID()
56: {
57:   char buf[BUFSIZE];
58:   SDMSERVICE service;
59:   SDMDATA data;
60:
61:   service.source = DataManager;
62:   service.source.setSensorID(1); //DataManager sensor_id is always 1
63:   service.destination.setPort(my_port);
64:   service.command_id = 264; //Could be queried for by using a SDMReqReg with item_name =
  ReturnSensorID
65:   service.length = 4;
66:   PUT_INT(service.data,PID);
67:   service.Send();
68:
69:   mm.BlockGetMessage(buf);
70:   data.Unmarshal(buf);
71:
72:   my_PID = GET_LONG(data.msg);
73:   my_sensorID = GET_LONG(&data.msg[4]); //Could be gotten from the Messagemanipulator
74:   printf("My PID is %ld \n",my_PID);
75:   printf("My sensor_id is %ld \n",my_sensorID);
76: }
77:

```

```
78: void CancelxTEDS()
79: {
80:   SDMCancelxTEDS cancel;
81:   printf("Canceling xTEDS \n");
82:   cancel.source.setSensorID(1);
83:   cancel.source.setPort(my_port);
84:   cancel.Send();
85: }
```


File: sdm/app/test/Makefile.uclinux

```
1: ifndef PETALINUX
2: $(error You must source the petalinux/settings.sh script before working with PetaLinux)
3: endif
4:
5: # Point to default PetaLinux root directory
6: ifndef ROOTDIR
7: ROOTDIR=$(PETALINUX)/software/petalinux-dist
8: endif
9:
10: PATH:=$(PATH):$(ROOTDIR)/tools
11:
12: UCLINUX_BUILD_USER = 1
13: -include $(ROOTDIR)/.config
14: -include $(ROOTDIR)/$(CONFIG_LINUXDIR)/.config
15: LIBCDIR = $(CONFIG_LIBCDIR)
16: -include $(ROOTDIR)/config.arch
17: ROMFSDIR=$(ROOTDIR)/romfs
18: ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh
19:
20: LDFLAGS+=-L../common
21: LDLIBS+=-lpthread -lsdm
22:
23: APPS = GetTimeTest
24:
25: # Add any other object files to this list below
26: APP_OBJS = TatTask.o
27:
28: all: $(APPS)
29:
30: GetTimeTest: GetTimeTest.o
31: $(CXX) $(LDFLAGS) -o $@ $^ $(LDLIBS)
32:
33: clean:
34: -rm -f $(APPS) *.elf *.gdb *.o
35:
36: romfs:
37: $(ROMFSINST) GetTimeTest /bin/GetTimeTest
38:
39: %.o: %.cpp
```

```
40: $(CXX) -c $(CXXFLAGS) -o $$@ $<
41:
42:
43: # Targets for the required .config files - if they don't exist, the tree isn't
44: # configured. Tell the user this, how to fix it, and exit.
45: ${ROOTDIR}/config.arch ${ROOTDIR}/.config:
46: @echo "Error: You must configure the PetaLinux tree before compiling your application"
47: @echo ""
48: @echo "Change directory to ../../petalinux-dist and 'make menuconfig' or 'make xconfig'"
49: @echo ""
50: @echo "Once the tree is configured, return to this directory, and re-run make."
51: @echo ""
52: @exit -1
53:
```

File: sdm/app/test/Makefile

```
1: include ../../Makefile.defs
2:
3: BUILDTARGETS=badxTEDS1 magnetometer_test magtest \
4:   ReqRegTest ReqxTEDStest ServicePID StressxTEDtest \
5:   badReqReg1 TestDMService TestRegEx testSDMTat \
6:   messagecountconsumer MessageLogAddAll MessageLogRemoveAll \
7:   xTEDSPoster SearchTest MessageClassTest ReqReg1.4Test DMServicesTest TMSubsTest
   TCPTester \
8:   SearchSubTest ModificationSearchSubTest MinimalxTEDSgetIP \
9:   FaultMsgtest BreadBoardTest VarInfoParserTest VarReq FileServiceTest DevicesTest
   RoboHubTest KillTester TatTest TaskPostTest GetTimeTest
10:
11: .PHONY: all clean distclean
12:
13: SUBDIRS=DMTests PMTests ClassTests
14:
15: all: $(BUILDTARGETS)
16: for dir in $(SUBDIRS); do \
17:     make -C $$dir; \
18: done
19:
20: DNET_IRU_400_test: DNET_IRU_400_test.o
21: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lsdm
22:
23: DNET_Reaction_Wheel_4NM_test: DNET_Reaction_Wheel_4NM_test.o
24: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lsdm
25:
26: badxTEDS1: badxTEDS1.o
27: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lsdm
28:
29: badReqReg1: badReqReg1.o
30: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lsdm
31:
32: ReqxTEDStest: ReqxTEDStest.o
33: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lsdm
34:
35: ReqRegTest: ReqRegTest.o
36: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lsdm
37:
```

38: magnetometer_test: magnetometer_test.o
39: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
40:
41: magtest: magtest.o
42: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
43:
44: StressxTEDtest: StressxTEDtest.o
45: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
46:
47: ServicePID: ServicePID.o
48: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
49:
50: TestDMService: TestDMService.o
51: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
52:
53: TestRegEx: TestRegEx.o
54: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
55:
56: testSDMTat: testSDMTat.o
57: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
58:
59: test2: test2.o
60: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
61:
62: messagecountconsumer: messagecountconsumer.o
63: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
64:
65: MessageLogAdd: MessageLogAdd.o
66: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
67:
68: MessageLogAddAll: MessageLogAddAll.o
69: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
70:
71: MessageLogRemove: MessageLogRemove.o
72: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
73:
74: MessageLogRemoveAll: MessageLogRemoveAll.o
75: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
76:
77: xTEDSPoster: xTEDSPoster.o
78: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM

79:
80: SearchTest: SearchTest.o
81: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
82:
83: MessageClassTest: MessageClassTest.o
84: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
85:
86: ReqReg1.4Test: ReqReg1.4Test.o
87: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
88:
89: DMServicesTest: DMServicesTest.o
90: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
91:
92: TMSubsTest: TMSubsTest.o
93: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
94:
95: TCPTester: TCPTester.o
96: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
97:
98: SearchSubTest: SearchSubTest.o
99: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
100:
101: BreadBoardTest: BreadBoardTest.o
102: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
103:
104: RoboHubTest: RoboHubTest.o
105: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
106:
107: ModificationSearchSubTest: ModificationSearchSubTest.o
108: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
109:
110: MinimalxTEDSgetIP: MinimalxTEDSgetIP.o
111: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
112:
113: FaultMsgtest: FaultMsgtest.o
114: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
115:
116: VarInfoParserTest: VarInfoParserTest.o
117: \$(CXX) \$(CXXFLAGS) -o \$@ \$^ \$(BOOSTFLAGS) -lpthread -L../common -lSDM
118:
119: VarReq: VarReq.o

```

120: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
121:
122: FileServiceTest: FileServiceTest.o
123: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
124:
125: DevicesTest: DevicesTest.o
126: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
127:
128: MessageManipulatorTest: MessageManipulatorTest.o
129: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
130:
131: KillTester: KillTester.o
132: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
133:
134: TatTest: TatTest.o
135: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
136:
137: TaskPostTest: TaskPostTest.o
138: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
139:
140: GetTimeTest: GetTimeTest.o
141: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../common -lSDM
142:
143: %.o: %.cpp
144: $(CXX) $(CXXFLAGS) -c $<
145:
146: clean:
147: rm -f *.o *~
148: for dir in $(SUBDIRS); do \
149:     make -C $$dir clean; \
150: done
151:
152: distclean: clean
153: rm -f $(BUILDTARGETS)
154: rm -f SDMMessages*log
155: for dir in $(SUBDIRS); do \
156:     make -C $$dir distclean; \
157: done
158:

```

File: sdm/app/test/testSDMTat.cpp

```
1: #include <iostream>
2: #include "../common/message/SDMTat.h"
3: #include "../common/message/SDMmessage.h"
4:
5: int main(int argc, char *argv[])
6: {
7:     SDMTat tat;
8:
9:     SDMInit(argc,argv);
10:    tat.seconds = 35;
11:    tat.useconds = 2;
12:    tat.destination.setSensorID(2);
13:    printf("Sending Tat \n");
14:    tat.Send();
15:    printf("Finished testSDMTat \n");
16: }
```

File: sdm/app/test/badReqReg1.cpp

```
1: #include "../common/message/SDMmessage.h"
2: #include "../common/message/SDMReqReg.h"
3: #include "../common/UDPcom.h"
4: #include "../common/message_defs.h"
5:
6: #include <stdio.h>
7: #include <unistd.h>
8: #include <stdlib.h>
9: #include <string.h>
10:
11: void nullrequest_test(void);
12: void badquallist_test(void);
13: void badreply(void);
14: void repeat_test(void);
15:
16: int main(int argc, char** argv)
17: {
18:     SDMInit(argc,argv);
19:     nullrequest_test();
20:     badquallist_test();
21:     badreply();
22:     repeat_test();
23: }
24:
25: //null request sends a SDMReqReg message with no itemname and no quallist
26: void nullrequest_test(void)
27: {
28:     SDMReqReg request;
29:     request.reply = SDM_REQREG_CURRENT;
30:     request.destination.setPort(0);
31:     request.id = 0;
32:     memset(request.item_name, '\0', 81);
33:     memset(request.quallist, '\0', 100);
34:     printf("Starting null request test \n");
35:     request.Send();
36:     printf("Test Finished \n");
37:     sleep(5);
38: }
39:
```



```

40: //badquallist sends a SDMReqReg message with bad quallists
41: void badquallist_test(void)
42: {
43:     SDMReqReg request;
44:     request.reply = SDM_REQREG_CURRENT;
45:     request.destination.setPort(0);
46:     request.id = 0;
47:     memset(request.item_name, '\0', 81);
48:     printf("Starting bad quallist test \n");
49:     strcpy(request.quallist, "HELLO WORLD!");
50:     request.Send();
51:     strcpy(request.quallist, "HELLO WORLD!/>");
52:     request.Send();
53:     strcpy(request.quallist, "<HELLO WORLD!/>");
54:     request.Send();
55:     strcpy(request.quallist, "</>");
56:     request.Send();
57:     for(int j=0;j<10;++j)
58:     {
59:         for(int i=0;i<100;++i)
60:             request.quallist[i] = rand() % 256;
61:         request.Send();
62:     }
63:     printf("Test Finished \n");
64:     sleep(5);
65: }
66:
67: //badreply tried all the possible reply values
68: void badreply(void)
69: {
70:     SDMReqReg request;
71:     request.destination.setPort(4000);
72:     strcpy(request.item_name, "Deregistration");
73:     strcpy(request.quallist, "</>");
74:     printf("Starting bad reply test \n");
75:     for(int i=0;i<256;++i)
76:     {
77:         request.reply = i;
78:         request.id = i;
79:         request.Send();
80:     }

```

```

81: printf("Test Finished \n");
82: sleep(5);
83: }
84:
85: //repeat test sends multiple copies of the same request thing to overflow a buffer
86: void repeat_test(void)
87: {
88:   SDMReqReg request;
89:   request.destination.setPort(4000);
90:   request.id = 1;
91:   request.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
92:   strcpy(request.item_name,"Deregistration");
93:   strcpy(request.quallist,"</>");
94:   printf("Starting repeat test \n");
95:   for(int i=0;i<50;++i)
96:     request.Send();
97:   printf("Test Finished \n");
98:   sleep(5);
99: }

```

File: sdm/app/test/ReqReg1.4Test.cpp

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <unistd.h>
4: #include <string.h>
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMRegInfo.h"
7: #include "../common/MessageManager/MessageManager.h"
8:
9: #define LOGMSGID 1
10:
11: int main(int argc, char* argv[])
12: {
13:     SDMInit(argc,argv);
14:     long my_port = getPort();
15:     MessageManager mm;
16:     bool cont = true;
17:     long result;
18:     char buf[BUFSIZE];
19:     long length = BUFSIZE;
20:
21:     mm.Async_Init(my_port);
22:     sleep(1);
23:     //_Test #1:_
24:
25:     SDMReqReg request;
26:     SDMRegInfo info;
27:     // Let's find out where the DM is running.
28:     strcpy(request.device, "DataManager");
29:     //strcpy(request.interface, "Message_Log");
30:     //strcpy(request.item_name, "Enable_Logging");
31:     strcpy(request.qualist, "");
32:     request.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
33:     request.destination.setPort(my_port);
34:     request.id = LOGMSGID;
35:     request.Send();
36:
37:     printf(" \n \n \nTest 1 \n");
38:     while(cont)
39: {
```

```

40:     if (mm.IsReady())
41:     {
42:         mm.GetMessage(buf,length);
43:         result = info.Unmarshal(buf);
44:         if(result == SDM_NO_FURTHER_DATA_PROVIDER)
45:             cont = false;
46:         if(cont == true)
47:         {
48:             memset(buf,0,BUFSIZE);
49:             if(info.MsgToString(buf,BUFSIZE) > 0)
50:                 printf("recv'd: %s \n",buf);
51:         }
52:     }
53: }
54:
55: // _Test #2: _
56: cont = true;
57: // Let's find out where the DM is running.
58: strcpy(request.device, "DataManager");
59: //strcpy(request.interface, "Message_Log");
60: strcpy(request.item_name, "Enable_Logging");
61: strcpy(request.quallist, "");
62: request.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
63: request.destination.setPort(my_port);
64: request.id = LOGMSGID;
65: request.Send();
66:
67: printf("\n \n \nTest 2 \n");
68: while(cont)
69: {
70:     if (mm.IsReady())
71:     {
72:         mm.GetMessage(buf,length);
73:         result = info.Unmarshal(buf);
74:         if(result == SDM_NO_FURTHER_DATA_PROVIDER)
75:             cont = false;
76:         if(cont == true)
77:         {
78:             memset(buf,0,BUFSIZE);
79:             if(info.MsgToString(buf,BUFSIZE) > 0)
80:                 printf("recv'd: %s \n",buf);

```

```

81:     }
82: }
83: }
84: // _Test #3: _
85: cont = true;
86: // Let's find out where the DM is running.
87: strcpy(request.device, "DataManager");
88: strcpy(request.interface, "Message_Log");
89: strcpy(request.item_name, "Enable_Logging");
90: strcpy(request.quallist, "");
91: request.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
92: request.destination.setPort(my_port);
93: request.id = LOGMSGID;
94: request.Send();
95:
96: printf(" \n \n \nTest 3 \n");
97: while(cont)
98: {
99:     if (mm.IsReady())
100:     {
101:         mm.GetMessage(buf,length);
102:         result = info.Unmarshal(buf);
103:         if(result == SDM_NO_FURTHER_DATA_PROVIDER)
104:             cont = false;
105:         if(cont == true)
106:         {
107:             memset(buf,0,BUFSIZE);
108:             if(info.MsgToString(buf,BUFSIZE) > 0)
109:                 printf("recv'd: %s \n",buf);
110:         }
111:     }
112: }
113:
114: // _Test #4: _
115: cont = true;
116: // Let's find out where the DM is running.
117: strcpy(request.device, "DataManager");
118: strcpy(request.interface, "Message_Log");
119: //strcpy(request.item_name, "Enable_Logging");
120: memset(request.item_name,0,33);
121: strcpy(request.quallist, "");

```

```

122: request.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
123: request.destination.setPort(my_port);
124: request.id = LOGMSGID;
125: request.Send();
126:
127: printf(" \n \n \nTest 4 \n");
128: while(cont)
129: {
130:     if (mm.IsReady())
131:     {
132:         mm.GetMessage(buf,length);
133:         result = info.Unmarshal(buf);
134:         if(result == SDM_NO_FURTHER_DATA_PROVIDER)
135:             cont = false;
136:         if(cont == true)
137:         {
138:             memset(buf,0,BUFSIZE);
139:             if(info.MsgToString(buf,BUFSIZE) > 0)
140:                 printf("recv'd: %s \n",buf);
141:         }
142:     }
143: }
144:
145: request.reply = SDM_REQREG_CANCEL;
146: request.Send();
147:
148: SDMReqReg reqreg;
149: reqreg.reply = SDM_REQREG_CANCEL;
150: reqreg.destination = request.destination;
151: reqreg.Send();
152: }

```

File: sdm/app/test/MessageClassTest.cpp

```
1: #include "../common/message/SDMAck.h"
2: #include "../common/message/SDMCancel.h"
3: #include "../common/message/SDMCancelxTEDS.h"
4: #include "../common/message/SDMCode.h"
5: #include "../common/message/SDMCommand.h"
6: #include "../common/message/SDMConsume.h"
7: #include "../common/message/SDMData.h"
8: #include "../common/message/SDMDeletesub.h"
9: #include "../common/message/SDMDMLLeader.h"
10: #include "../common/message/SDMElection.h"
11: #include "../common/message/SDMError.h"
12: #include "../common/message/SDMHeartbeat.h"
13: #include "../common/message/SDMKill.h"
14: #include "../common/message/SDMPostTask.h"
15: #include "../common/message/SDMReady.h"
16: #include "../common/message/SDMRegInfo.h"
17: #include "../common/message/SDMRegPM.h"
18: #include "../common/message/SDMReqCode.h"
19: #include "../common/message/SDMReqReg.h"
20: #include "../common/message/SDMReqxTEDS.h"
21: #include "../common/message/SDMSearch.h"
22: #include "../common/message/SDMSearchReply.h"
23: #include "../common/message/SDMSerreqst.h"
24: #include "../common/message/SDMService.h"
25: #include "../common/message/SDMSubreqst.h"
26: #include "../common/message/SDMTask.h"
27: #include "../common/message/SDMTaskFinished.h"
28: #include "../common/message/SDMTat.h"
29: #include "../common/message/SDMVarInfo.h"
30: #include "../common/message/SDMVarReq.h"
31: #include "../common/message/SDMxTEDS.h"
32: #include "../common/message/SDMxTEDSInfo.h"
33: // #include "../common/message/SDMxTEDSUpdate.h"
34:
35:
36: #include <stdio.h>
37: #include <stdlib.h>
38: #include <string.h>
39:
```

```

40: int main(void)
41: {
42: char buf[BUFSIZE];
43: int marshalsize = 0;
44: int unmarshalsize = 0;
45:
46: SDMAck ack;
47: marshalsize = ack.Marshal(buf);
48: unmarshalsize = ack.Unmarshal(buf);
49: if(unmarshalsize == SDM_INVALID_MESSAGE)
50:     printf("SDMAck invalid message \n");
51: if(marshalsize != unmarshalsize)
52:     printf("SDMAck marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
53: memset(buf,0,BUFSIZE);
54:
55: SDMCancel cancel;
56: marshalsize = cancel.Marshal(buf);
57: unmarshalsize = cancel.Unmarshal(buf);
58: if(unmarshalsize == SDM_INVALID_MESSAGE)
59:     printf("SDMCancel invalid message \n");
60: if(marshalsize != unmarshalsize)
61:     printf("SDMCancel marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
62: memset(buf,0,BUFSIZE);
63:
64: SDMCancelxTEDS cancelxTEDS;
65: marshalsize = cancelxTEDS.Marshal(buf);
66: unmarshalsize = cancelxTEDS.Unmarshal(buf);
67: if(unmarshalsize == SDM_INVALID_MESSAGE)
68:     printf("SDMCancelxTEDS invalid message \n");
69: if(marshalsize != unmarshalsize)
70:     printf("SDMCancelxTEDS marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
71: memset(buf,0,BUFSIZE);
72:
73: SDMCode code;
74: marshalsize = code.Marshal(buf);
75: unmarshalsize = code.Unmarshal(buf);
76: if(unmarshalsize == SDM_INVALID_MESSAGE)
77:     printf("SDMCode invalid message \n");
78: if(marshalsize != unmarshalsize)

```



```

79:  printf("SDMCode marshal:  %d,  unmarshal:  %d  return  sizes  do  not  match
\n",marshalsize,unmarshalsize);
80:  memset(buf,0,BUFSIZE);
81:
82:  SDMCommand command;
83:  marshalsize = command.Marshal(buf);
84:  unmarshalsize = command.Unmarshal(buf);
85:  if(unmarshalsize == SDM_INVALID_MESSAGE)
86:      printf("SDMCommand invalid message \n");
87:  if(marshalsize != unmarshalsize)
88:      printf("SDMCommand marshal:  %d,  unmarshal:  %d  return  sizes  do  not  match
\n",marshalsize,unmarshalsize);
89:  memset(buf,0,BUFSIZE);
90:
91:  SDMConsume consume;
92:  marshalsize = consume.Marshal(buf);
93:  unmarshalsize = consume.Unmarshal(buf);
94:  if(unmarshalsize == SDM_INVALID_MESSAGE)
95:      printf("SDMConsume invalid message \n");
96:  if(marshalsize != unmarshalsize)
97:      printf("SDMConsume marshal:  %d,  unmarshal:  %d  return  sizes  do  not  match
\n",marshalsize,unmarshalsize);
98:  memset(buf,0,BUFSIZE);
99:
100:  SDMDData data;
101:  marshalsize = data.Marshal(buf);
102:  unmarshalsize = data.Unmarshal(buf);
103:  if(unmarshalsize == SDM_INVALID_MESSAGE)
104:      printf("SDMDData invalid message \n");
105:  if(marshalsize != unmarshalsize)
106:      printf("SDMDData marshal:  %d,  unmarshal:  %d  return  sizes  do  not  match
\n",marshalsize,unmarshalsize);
107:  memset(buf,0,BUFSIZE);
108:
109:  SDMDeletesub deletesub;
110:  marshalsize = deletesub.Marshal(buf);
111:  unmarshalsize = deletesub.Unmarshal(buf);
112:  if(unmarshalsize == SDM_INVALID_MESSAGE)
113:      printf("SDMDeletesub invalid message \n");
114:  if(marshalsize != unmarshalsize)
115:      printf("SDMDeletesub marshal:  %d,  unmarshal:  %d  return  sizes  do  not  match
\n",marshalsize,unmarshalsize);

```

```

116:  memset(buf,0,BUFSIZE);
117:
118:  SDMDMLeader leader;
119:  marshalsize = leader.Marshal(buf);
120:  unmarshalsize = leader.Unmarshal(buf);
121:  if(unmarshalsize == SDM_INVALID_MESSAGE)
122:      printf("SDMDMLeader invalid message \n");
123:  if(marshalsize != unmarshalsize)
124:      printf("SDMDMLeader marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
125:  memset(buf,0,BUFSIZE);
126:
127:  SDMElection election;
128:  marshalsize = election.Marshal(buf);
129:  unmarshalsize = election.Unmarshal(buf);
130:  if(unmarshalsize == SDM_INVALID_MESSAGE)
131:      printf("SDMElection invalid message \n");
132:  if(marshalsize != unmarshalsize)
133:      printf("SDMElection marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
134:  memset(buf,0,BUFSIZE);
135:
136:  SDMEError error;
137:  marshalsize = error.Marshal(buf);
138:  unmarshalsize = error.Unmarshal(buf);
139:  if(unmarshalsize == SDM_INVALID_MESSAGE)
140:      printf("SDMEError invalid message \n");
141:  if(marshalsize != unmarshalsize)
142:      printf("SDMEError marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
143:  memset(buf,0,BUFSIZE);
144:
145:  SDMHeartbeat heartbeat;
146:  marshalsize = heartbeat.Marshal(buf);
147:  unmarshalsize = heartbeat.Unmarshal(buf);
148:  if(unmarshalsize == SDM_INVALID_MESSAGE)
149:      printf("SDMHeartbeat invalid message \n");
150:  if(marshalsize != unmarshalsize)
151:      printf("SDMHeartbeat marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
152:  memset(buf,0,BUFSIZE);
153:

```

```

154:   SDMKill kill;
155:   marshalsize = kill.Marshal(buf);
156:   unmarshalsize = kill.Unmarshal(buf);
157:   if(unmarshalsize == SDM_INVALID_MESSAGE)
158:       printf("SDMKill invalid message \n");
159:   if(marshalsize != unmarshalsize)
160:       printf("SDMKill marshal:  %d,  unmarshal:  %d  return sizes do not match
\n",marshalsize,unmarshalsize);
161:   memset(buf,0,BUFSIZE);
162:
163:   SDMPostTask posttask;
164:   marshalsize = posttask.Marshal(buf);
165:   unmarshalsize = posttask.Unmarshal(buf);
166:   if(unmarshalsize == SDM_INVALID_MESSAGE)
167:       printf("SDMPostTask invalid message \n");
168:   if(marshalsize != unmarshalsize)
169:       printf("SDMPostTask marshal:  %d,  unmarshal:  %d  return sizes do not match
\n",marshalsize,unmarshalsize);
170:   memset(buf,0,BUFSIZE);
171:
172:   SDMReady ready;
173:   marshalsize = ready.Marshal(buf);
174:   unmarshalsize = ready.Unmarshal(buf);
175:   if(unmarshalsize == SDM_INVALID_MESSAGE)
176:       printf("SDMReady invalid message \n");
177:   if(marshalsize != unmarshalsize)
178:       printf("SDMReady marshal:  %d,  unmarshal:  %d  return sizes do not match
\n",marshalsize,unmarshalsize);
179:   memset(buf,0,BUFSIZE);
180:
181:   SDMRegInfo reginfo;
182:   marshalsize = reginfo.Marshal(buf);
183:   unmarshalsize = reginfo.Unmarshal(buf);
184:   if(unmarshalsize == SDM_INVALID_MESSAGE)
185:       printf("SDMRegInfo invalid message \n");
186:   if(marshalsize != unmarshalsize)
187:       printf("SDMRegInfo marshal:  %d,  unmarshal:  %d  return sizes do not match
\n",marshalsize,unmarshalsize);
188:   memset(buf,0,BUFSIZE);
189:
190:   SDMRegPM regpm;
191:   marshalsize = regpm.Marshal(buf);

```

```

192:  unmarshalsize = regpm.Unmarshal(buf);
193:  if(unmarshalsize == SDM_INVALID_MESSAGE)
194:      printf("SDMRegPM invalid message \n");
195:  if(marshalsize != unmarshalsize)
196:      printf("SDMRegPM marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
197:  memset(buf,0,BUFSIZE);
198:
199:  SDMReqCode reqcode;
200:  marshalsize = reqcode.Marshal(buf);
201:  unmarshalsize = reqcode.Unmarshal(buf);
202:  if(unmarshalsize == SDM_INVALID_MESSAGE)
203:      printf("SDMReqCode invalid message \n");
204:  if(marshalsize != unmarshalsize)
205:      printf("SDMReqCode marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
206:  memset(buf,0,BUFSIZE);
207:
208:  SDMReqReg reqreg;
209:  marshalsize = reqreg.Marshal(buf);
210:  unmarshalsize = reqreg.Unmarshal(buf);
211:  if(unmarshalsize == SDM_INVALID_MESSAGE)
212:      printf("SDMReqReg invalid message \n");
213:  if(marshalsize != unmarshalsize)
214:      printf("SDMReqReg marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
215:  memset(buf,0,BUFSIZE);
216:
217:  SDMReqxTEDS reqxteds;
218:  marshalsize = reqxteds.Marshal(buf);
219:  if(marshalsize > 0)
220:      printf("SDMReqxTEDS no select marshal succeeded when it should fail \n");
221:  reqxteds.select = 0;
222:  memset(buf,0,BUFSIZE);
223:  marshalsize = reqxteds.Marshal(buf);
224:  unmarshalsize = reqxteds.Unmarshal(buf);
225:  if(unmarshalsize == SDM_INVALID_MESSAGE)
226:      printf("SDMReqxTEDS select=0 invalid message \n");
227:  if(marshalsize != unmarshalsize)
228:      printf("SDMReqxTEDS select=0 marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
229:  memset(buf,0,BUFSIZE);

```

```

230: reqxteds.select = 1;
231: marshalsize = reqxteds.Marshal(buf);
232: unmarshalsize = reqxteds.Unmarshal(buf);
233: if(unmarshalsize == SDM_INVALID_MESSAGE)
234:     printf("SDMReqxTEDS select=1 invalid message \n");
235: if(marshalsize != unmarshalsize)
236:     printf("SDMReqxTEDS select=1 marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
237: memset(buf,0,BUFSIZE);
238: reqxteds.select = 2;
239: marshalsize = reqxteds.Marshal(buf);
240: unmarshalsize = reqxteds.Unmarshal(buf);
241: if(unmarshalsize == SDM_INVALID_MESSAGE)
242:     printf("SDMReqxTEDS select=2 invalid message \n");
243: if(marshalsize != unmarshalsize)
244:     printf("SDMReqxTEDS select=2 marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
245: memset(buf,0,BUFSIZE);
246: reqxteds.select = 3;
247: marshalsize = reqxteds.Marshal(buf);
248: unmarshalsize = reqxteds.Unmarshal(buf);
249: if(unmarshalsize == SDM_INVALID_MESSAGE)
250:     printf("SDMReqxTEDS select=3 invalid message \n");
251: if(marshalsize != unmarshalsize)
252:     printf("SDMReqxTEDS select=3 marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
253: memset(buf,0,BUFSIZE);
254:
255: SDMSearch search;
256: marshalsize = search.Marshal(buf);
257: unmarshalsize = search.Unmarshal(buf);
258: if(unmarshalsize == SDM_INVALID_MESSAGE)
259:     printf("SDMSearch invalid message \n");
260: if(marshalsize != unmarshalsize)
261:     printf("SDMSearch marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
262: memset(buf,0,BUFSIZE);
263:
264: SDMSearchReply searchreply;
265: marshalsize = searchreply.Marshal(buf);
266: unmarshalsize = searchreply.Unmarshal(buf);
267: if(unmarshalsize == SDM_INVALID_MESSAGE)

```

```

268:     printf("SDMSearchReply invalid message \n");
269:     if(marshalsize != unmarshalsize)
270:         printf("SDMSearchReply marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
271:     memset(buf,0,BUFSIZE);
272:
273:     SDMSerreqst serreqst;
274:     marshalsize = serreqst.Marshal(buf);
275:     unmarshalsize = serreqst.Unmarshal(buf);
276:     if(unmarshalsize == SDM_INVALID_MESSAGE)
277:         printf("SDMSerreqst invalid message \n");
278:     if(marshalsize != unmarshalsize)
279:         printf("SDMSerreqst marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
280:     memset(buf,0,BUFSIZE);
281:
282:     SDMService service;
283:     marshalsize = service.Marshal(buf);
284:     unmarshalsize = service.Unmarshal(buf);
285:     if(unmarshalsize == SDM_INVALID_MESSAGE)
286:         printf("SDMService invalid message \n");
287:     if(marshalsize != unmarshalsize)
288:         printf("SDMService marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
289:     memset(buf,0,BUFSIZE);
290:
291:     SDMSubreqst subreqst;
292:     marshalsize = subreqst.Marshal(buf);
293:     unmarshalsize = subreqst.Unmarshal(buf);
294:     if(unmarshalsize == SDM_INVALID_MESSAGE)
295:         printf("SDMSubreqst invalid message \n");
296:     if(marshalsize != unmarshalsize)
297:         printf("SDMSubreqst marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
298:     memset(buf,0,BUFSIZE);
299:
300:     SDMTask task;
301:     marshalsize = task.Marshal(buf);
302:     unmarshalsize = task.Unmarshal(buf);
303:     if(unmarshalsize == SDM_INVALID_MESSAGE)
304:         printf("SDMTask invalid message \n");
305:     if(marshalsize != unmarshalsize)

```

```

306:     printf("SDMTask marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
307:     memset(buf,0,BUFSIZE);
308:
309:     SDMTaskFinished taskfinished;
310:     marshalsize = taskfinished.Marshal(buf);
311:     unmarshalsize = taskfinished.Unmarshal(buf);
312:     if(unmarshalsize == SDM_INVALID_MESSAGE)
313:         printf("SDMTaskFinished invalid message \n");
314:     if(marshalsize != unmarshalsize)
315:         printf("SDMTaskFinished marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
316:     memset(buf,0,BUFSIZE);
317:
318:     SDMTat tat;
319:     marshalsize = tat.Marshal(buf);
320:     unmarshalsize = tat.Unmarshal(buf);
321:     if(unmarshalsize == SDM_INVALID_MESSAGE)
322:         printf("SDMTat invalid message \n");
323:     if(marshalsize != unmarshalsize)
324:         printf("SDMTat marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
325:     memset(buf,0,BUFSIZE);
326:
327:     SDMVarInfo varinfo;
328:     marshalsize = varinfo.Marshal(buf);
329:     unmarshalsize = varinfo.Unmarshal(buf);
330:     if(unmarshalsize == SDM_INVALID_MESSAGE)
331:         printf("SDMVarInfo invalid message \n");
332:     if(marshalsize != unmarshalsize)
333:         printf("SDMVarInfo marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
334:     memset(buf,0,BUFSIZE);
335:
336:     SDMVarReq varreq;
337:     marshalsize = varreq.Marshal(buf);
338:     unmarshalsize = varreq.Unmarshal(buf);
339:     if(unmarshalsize == SDM_INVALID_MESSAGE)
340:         printf("SDMVarReq invalid message \n");
341:     if(marshalsize != unmarshalsize)
342:         printf("SDMVarReq marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);

```

```

343:  memset(buf,0,BUFSIZE);
344:
345:  SDMxTEDS xteds;
346:  marshalsize = xteds.Marshal(buf);
347:  unmarshalsize = xteds.Unmarshal(buf);
348:  if(unmarshalsize == SDM_INVALID_MESSAGE)
349:      printf("SDMxTEDS invalid message \n");
350:  if(marshalsize != unmarshalsize)
351:      printf("SDMxTEDS marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
352:  memset(buf,0,BUFSIZE);
353:
354:  SDMxTEDSInfo xtedsinfo;
355:  marshalsize = xtedsinfo.Marshal(buf);
356:  unmarshalsize = xtedsinfo.Unmarshal(buf);
357:  if(unmarshalsize == SDM_INVALID_MESSAGE)
358:      printf("SDMxTEDSInfo invalid message \n");
359:  if(marshalsize != unmarshalsize)
360:      printf("SDMxTEDSInfo marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
361:  memset(buf,0,BUFSIZE);
362:
363:  /*SDMxTEDSUpdate update;
364:  marshalsize = update.Marshal(buf);
365:  unmarshalsize = update.Unmarshal(buf);
366:  if(unmarshalsize == SDM_INVALID_MESSAGE)
367:      printf("SDMxTEDSUpdate invalid message \n");
368:  if(marshalsize != unmarshalsize)
369:      printf("SDMxTEDSUpdate marshal: %d, unmarshal: %d return sizes do not match
\n",marshalsize,unmarshalsize);
370:  memset(buf,0,BUFSIZE);*/
371: }

```


File: sdm/app/test/TestDMService.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMCancelxTEDS.h"
3: #include "../common/message/SDMService.h"
4: #include "../common/message/SDMData.h"
5: #include "../common/message/SDMConsume.h"
6: #include "../common/MessageManager/MessageManager.h"
7: #include <string.h>
8: #include <sys/types.h>
9: #include <sys/stat.h>
10: #include <fcntl.h>
11: #include <unistd.h>
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <pthread.h>
15: #include <sys/socket.h>
16: #include <netinet/in.h>
17: #include <arpa/inet.h>
18:
19: void RegisterxTEDS();
20: void CancelxTEDS();
21: void* Publisher(void *);
22: void* Listener(void *);
23:
24: long my_port;
25:
26: int main(int argc, char** argv)
27: {
28:     SDMService service;
29:     SDMConsume con;
30:     MessageManager mm;
31:     char buf[BUFSIZE];
32:     char name[128];
33:     char ip[17];
34:     long port;
35:     SDMComponent_ID id;
36:
37:     int pid = 0;
38:     SDMData dat;
39:
```

```

40: SDMInit(argc,argv);
41: my_port = getPort();
42: if (my_port < 0)
43: {
44:     printf("Error getting port. \n");
45:     return -1;
46: }
47: mm.Async_Init(my_port);
48:
49: RegisterxTEDS();
50: usleep(100);
51: service.source.setSensorID(1);
52: service.command_id = 264;
53: service.destination.setPort(my_port);
54: service.length = 4;
55: pid = atoi(argv[3]);
56: memcpy(service.data,&pid,4);
57: service.Send();
58:
59: if (mm.BlockGetMessage(buf) != SDM_Data)
60: {
61:     printf("Error, unknown message received. \n");
62:     return -1;
63: }
64: dat.Unmarshal(buf);
65:
66: pid = GET_LONG(dat.msg);
67: printf("My SensorID is %d \n",pid);
68: service.command_id = 262;
69: memcpy(service.data,&pid,4);
70: service.Send();
71:
72: if (mm.BlockGetMessage(buf) != SDM_Data)
73: {
74:     printf("Error, unknown message received. \n");
75:     return -1;
76: }
77: dat.Unmarshal(buf);
78:
79: memset(name,0,128);
80: strncpy(name,dat.msg,127);

```

```

81: printf("My name is %s \n",name);
82: service.command_id = 268;
83: service.Send();
84:
85: if (mm.BlockGetMessage(buf) != SDM_Data)
86: {
87:     printf("Error, unknown message received. \n");
88:     return -1;
89: }
90: dat.Unmarshal(buf);
91:
92: port = GET_LONG(dat.msg);
93: strcpy(ip,&dat.msg[4]);
94: printf("My port is %ld and my ip is %s \n",port,ip);
95: service.command_id = 267;
96: service.Send();
97: service.command_id = 270;
98: id.setSensorID(pid);
99: id.setPort(port);
100: id.setAddress(inet_addr(ip));
101: id.Unmarshal(service.data,0);
102: service.Send();
103:
104: if (mm.BlockGetMessage(buf) != SDM_Data)
105: {
106:     printf("Error, unknown message received. \n");
107:     return -1;
108: }
109: dat.Unmarshal(buf);
110:
111: printf("My componentKey is %s \n",dat.msg);
112: CancelxTEDS();
113: }
114:
115: void RegisterxTEDS()
116: {
117:     SDMxTEDS xteds;// create an xTEDS registration message
118:
119:     strcpy (xteds.xTEDS,"<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS version= \"2.0
\" name= \"Producer_xTEDS \"> \n \t<Application name= \"producer \" kind= \"data \" componentKey=
\"ProducerKey \"/> \n \t<Interface name= \"Producer_Interface \" id= \"1 \"> \n \t<Variable name= \"data
\" format= \"UINT16 \" kind= \"data \" qualifier= \"Single Value \"/> \n \t<Notification> \n \t \t<DataMsg

```

```

name= \"all \" id= \"1 \" msgArrival= \"PERIODIC \" msgRate= \"1 \"> \n \t \t <VariableRef name=
\"data \"/> \n \t \t </DataMsg> \n \t </Notification> \n \t </Interface> \n</xTEDS> \n\")); // set xTEDS
120:
121:  xteds.source.setSensorID(1);          // set the id of this application
122:  xteds.source.setPort(my_port);
123:  printf("Registering producer xTEDS on port %ld \n",my_port);
124:  xteds.Send();                          // register with the SDM
125: }
126:
127: void CancelxTEDS()
128: {
129:  SDMCancelxTEDS cancel;
130:  printf("Canceling xTEDS \n");
131:  cancel.source.setSensorID(1);
132:  cancel.source.setPort(my_port);
133:  cancel.Send();
134: }

```

File: sdm/app/test/DMServicesTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <unistd.h>
4: #include <vector>
5: #include <signal.h>
6:
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/message/SDMReqReg.h"
9: #include "../common/message/SDMConsume.h"
10: #include "../common/message/SDMCancel.h"
11: #include "../common/message/SDMData.h"
12: #include "../common/message/SDMCommand.h"
13: #include "../common/message/SDMService.h"
14: #include "../common/message/SDMSerreqst.h"
15: #include "../common/message/SDMMessage_ID.h"
16: #include "../common/MessageManager/MessageManager.h"
17:
18: #define DMSUB_REGISTRATION 1
19: #define DMSUB_DEREGISTRATION 2
20: #define DMSUB_REGCHANGE 3
21: #define DMCOM_CONVERTEDNAME 4
22: #define DMSER_SPANODE 5
23:
24: using namespace std;
25:
26: SDMMessage_ID registration_id;
27: SDMMessage_ID deregistration_id;
28: SDMMessage_ID regchange_id;
29: SDMMessage_ID send_sid;
30: SDMMessage_ID spanode;
31: SDMMessage_ID converted_spanode(1, 7);
32: SDMMessage_ID converted_name(1,5);
33: SDMMessage_ID converted_ip(1, 10);
34: SDMMessage_ID sensor_toip(1, 12);
35: SDMMessage_ID pid_tosid(1, 8);
36: SDMMessage_ID pid_tosidreply(1, 9);
37: SDMService sensor_id_msg;
38:
39: const int my_port = 4999;
```

```

40: bool providers_found = false;
41: vector<long> sensor_ids;
42: vector<SDMComponent_ID> comp_ids;
43: int signals_recld;
44: bool convert_name_ready = false;
45: //Tests whether the given argument is contained within sensor_ids
46: bool sensor_ids_contains(long num)
47: {
48:     for (unsigned int i = 0; i < sensor_ids.size(); i++)
49:     {
50:         if (sensor_ids[i] == num)
51:             return true;
52:     }
53:     return false;
54: }
55: bool comp_ids_contains(SDMComponent_ID id)
56: {
57:     for (unsigned int i = 0; i < comp_ids.size(); i++)
58:     {
59:         if (comp_ids[i] == id)
60:             return true;
61:     }
62:     return false;
63: }
64:
65:
66: void RegInfoHandler(SDMRegInfo reg)
67: {
68:     SDMConsume consume;
69:     consume.destination.setPort(my_port);
70:     consume.source = reg.source;
71:     consume.msg_id = reg.msg_id;
72:
73:     switch(reg.id)
74:     {
75:         case DMSUB_REGISTRATION:
76:             if (reg.source.getSensorID() == 1)
77:             {
78:                 printf("Data manager's Registration subscription found, subscribing... \n");
79:                 registration_id = reg.msg_id;
80:                 if (!comp_ids_contains(reg.source))

```

```

81:         comp_ids.push_back(reg.source);
82:         consume.Send();
83:         providers_found = true;
84:     }
85:     break;
86: case DMSUB_DEREGISTRATION:
87:     if (reg.source.getSensorID() == 1)
88:     {
89:         printf("Data manager's Deregistration subscription found, subscribing... \n");
90:         deregistration_id = reg.msg_id;
91:         if (!comp_ids_contains(reg.source))
92:             comp_ids.push_back(reg.source);
93:         consume.Send();
94:         providers_found = true;
95:     }
96:     break;
97: case DMSUB_REGCHANGE:
98:     if (reg.source.getSensorID() == 1)
99:     {
100:         printf("Data manager's Registration Change subscription found, subscribing... \n");
101:         regchange_id = reg.msg_id;
102:         if (!comp_ids_contains(reg.source))
103:             comp_ids.push_back(reg.source);
104:         consume.Send();
105:         providers_found = true;
106:     }
107:     break;
108: case DMCOM_CONVERTEDNAME:
109:     if (reg.source.getSensorID() == 1)
110:     {
111:         printf("Found DM convert name service. \n");
112:         if (reg.msg_id.getMessage() == 5)
113:         {
114:             send_sid = reg.msg_id;
115:             sensor_id_msg.source = reg.source;
116:             sensor_id_msg.command_id = reg.msg_id;
117:             sensor_id_msg.command_id.setMessage(6);
118:             sensor_id_msg.length = 4;
119:             sensor_id_msg.destination.setPort(my_port);
120:             convert_name_ready = true;
121:         }

```

```

122:         else if (reg.msg_id.getMessage() == 5)
123:         {
124:             converted_name = reg.msg_id;
125:         }
126:     }
127:     break;
128:     case DMSER_SPANODE:
129:         printf("Found service for SPANode. \n");
130:         spanode = reg.msg_id;
131:         break;
132:     default:
133:         printf("Received unexpected Reg Info type. \n");
134:     }
135: }
136:
137: void SendRegReqs()
138: {
139:     SDMReqReg request;
140:     request.destination.setPort(my_port);
141:     strcpy(request.interface, "DM_Interface");
142:
143:     printf("Sending request for the Registration subscription. \n");
144:     request.id = DMSUB_REGISTRATION;
145:     strcpy(request.item_name, "Registration");
146:     request.Send();
147:
148:     printf("Sending request for the Deregistration subscription. \n");
149:     request.id = DMSUB_DEREGISTRATION;
150:     strcpy(request.item_name, "Deregistration");
151:     request.Send();
152:
153:     printf("Sending request for the RegisterChange subscription. \n");
154:     request.id = DMSUB_REGCHANGE;
155:     strcpy(request.item_name, "RegisterChange");
156:     request.Send();
157:
158:     printf("Sending request for ConvertedDeviceName command. \n \n");
159:     strcpy(request.item_name, "ConvertedDeviceName");
160:     request.id = DMCOM_CONVERTEDNAME;
161:     request.Send();
162:

```



```

163:     printf("Sending request for SensorIDtoSPANode service. \n \n");
164:     strcpy(request.item_name, "SensorIDtoSPANode");
165:     request.id = DMSER_SPANODE;
166:     request.Send();
167: }
168:
169: void DataHandler(SDMMData data)
170: {
171:     unsigned long sensor_id;
172:     unsigned long address;
173:     unsigned short port;
174:     unsigned char type;
175:     char device_name[81];
176:
177:     if (data.msg_id == registration_id)
178:     {
179:         sensor_id = GET_INT(data.msg);
180:         printf("Registration data message received. Sensor id registered is %ld. \n", sensor_id);
181:         if (!sensor_ids_contains(sensor_id))
182:             sensor_ids.push_back(sensor_id);
183:         printf(" getting SPANode. \n");
184:
185:         SDMSERVICE ser;
186:         ser.source = data.source;
187:         ser.command_id = spanode;
188:         PUT_UINT(ser.data, sensor_id);
189:         ser.length = 4;
190:         ser.destination.setPort(my_port);
191:         ser.Send();
192:     }
193:     else if (data.msg_id == converted_spanode)
194:     {
195:         sensor_id = GET_UINT(data.msg);
196:
197:         printf(" SpaNode for sensor id %u is \"%s\" \n", sensor_id, data.msg + 4);
198:     }
199:     else if (data.msg_id == deregistration_id)
200:     {
201:         sensor_id = GET_INT(data.msg);
202:         printf("Deregistration data message received. Sensor id deregistered is %ld. \n", sensor_id);
203:         if (!sensor_ids_contains(sensor_id))

```

```

204:         sensor_ids.push_back(sensor_id);
205:     }
206:     else if (data.msg_id == regchange_id)
207:     {
208:         type = GET_UCHAR(data.msg);
209:         sensor_id = GET_INT(data.msg+1);
210:         if (type == 1)
211:             printf("Change registration data message received.  Sensor id %ld registered. \n",
sensor_id);
212:         else if (type == 2)
213:             printf("Change registration data message received.  Sensor id %ld deregistered. \n",
sensor_id);
214:         if (!sensor_ids_contains(sensor_id))
215:             sensor_ids.push_back(sensor_id);
216:     }
217:     else if (data.msg_id == converted_name)
218:     {
219:         sensor_id = GET_UINT(data.msg);
220:         memcpy(device_name, data.msg+4, 81);
221:         printf("The converted name for sensor id %ld is %s. \n", sensor_id, device_name);
222:     }
223:     else if (data.msg_id == converted_ip)
224:     {
225:         sensor_id = GET_UINT(data.msg);
226:         address = GET_UINT(data.msg+4);
227:         port = GET_USHORT(data.msg+8);
228:         printf("The ip address for sensor id %ld is %lu:%hd. \n",sensor_id,address,port);
229:     }
230:     else if (data.msg_id == pid_tosidreply)
231:     {
232:         sensor_id = GET_UINT(data.msg + 4);
233:         printf("PID TO SENSOR ID RETURNED %ld \n", sensor_id);
234:     }
235:
236:     static int count = 0;
237:     if ( count == 0 )
238:     {
239:         SDMSERVICE ser;
240:         ser.source = data.source;
241:         ser.command_id = pid_tosid;
242:         unsigned int mypid = 123;

```

```

243:     PUT_UINT(ser.data, mypid);
244:     ser.length = 4;
245:     ser.destination.setPort(my_port);
246:     ser.Send();
247: }
248: if (count ++ == 3)
249:     count = 0;
250: }
251:
252: void SigHandler(int sig_num)
253: {
254:     SDMCancel cancel;
255:     cancel.destination.setPort(my_port);
256:     if (sig_num == SIGINT)
257:     {
258:         signals_recd++;
259:         if (signals_recd == 1)        //Cancel the subscriptions, and prepare to send commands
260:         {
261:             printf(" \n \nCanceling subscriptions...");
262:             for (unsigned int i = 0; i < comp_ids.size(); i++)
263:             {
264:                 cancel.source = comp_ids[i];
265:                 cancel.msg_id = registration_id;
266:                 cancel.Send();
267:                 cancel.msg_id = deregistration_id;
268:                 cancel.Send();
269:                 cancel.msg_id = regchange_id;
270:                 cancel.Send();
271:             }
272:             printf("Done. \n");
273:             if (!convert_name_ready)
274:             {
275:                 printf("No RegInfo received for the convert name message. \n");
276:                 return;
277:             }
278:             printf(" \n");
279:             for (unsigned int i = 0; i < sensor_ids.size(); i++)
280:             {
281:                 printf("Requesting to convert sensor id %ld to device name... \n", sensor_ids[i]);
282:                 PUT_UINT(&sensor_id_msg.data, sensor_ids[i]);
283:                 sensor_id_msg.Send();

```

```

284:     }
285:     sensor_id_msg.command_id = sensor_toip;
286:     printf(" \n");
287:     for (unsigned int i = 0; i < sensor_ids.size(); i++)
288:     {
289:         printf("Requesting to convert sensor id %ld to ip... \n", sensor_ids[i]);
290:         PUT_UINT(&sensor_id_msg.data, sensor_ids[i]);
291:         sensor_id_msg.Send();
292:     }
293:     printf(" \n");
294:
295:     }
296:     else
297:         exit(EXIT_SUCCESS);
298: }
299: }
300: int main (int argc, char ** argv)
301: {
302:     char buf[BUFSIZE];
303:     long length;
304:     SDMRegInfo info;
305:     SDMDData data;
306:     SDMInit(argc, argv);
307:
308:     MessageManager mm;
309:     mm.Async_Init(my_port);
310:     signal(SIGINT, SigHandler);
311:
312:     printf("This program tests the Data Manager's  \"Registration \",  \"Deregistration \", and
\"Register Change \" subscriptions. It also tests the services  \"SendSensorID/ConvertedDeviceName \"
and  \"SensorIDtoIP/ConvertedIP \". After testing for registration/deregistrations, type CTRL+C to
convert all received sensor IDs to their device names and ip addresses. \n \n");
313:     sleep(3);
314:     while (1)
315:     {
316:         if (mm.IsReady())
317:         {
318:             switch(mm.GetMessage(buf, length))
319:             {
320:                 case SDM_RegInfo:
321:                     if (info.Unmarshal(buf) != SDM_NO_FURTHER_DATA_PROVIDER)
322:                     {

```

```

323:             RegInfoHandler(info);
324:         }
325:         break;
326:     case SDM_Data:
327:         data.Unmarshal(buf);
328:         DataHandler(data);
329:         break;
330:
331:     }
332:
333: }
334: else if (!providers_found)
335: {
336:     SendRegReqs();
337:     sleep(1);
338: }
339: else
340: {
341:     usleep(5000);
342: }
343: }
344:
345:
346: return 0;
347: }

```

File: sdm/app/test/GetTimeTest.cpp

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <sys/types.h>
4: #include <sys/stat.h>
5: #include <fcntl.h>
6: #include <unistd.h>
7: #include <string.h>
8:
9: #include "../common/Time/SDMTime.h"
10:
11: const char* g_strLogFileName = "GetTimeTest.log";
12:
13: int OpenLogFile();
14: int CloseLogFile();
15:
16: int main (int argc, char** argv)
17: {
18:     int iFileFd = 0;
19:
20:     unsigned int uiSeconds = 0, uiUSeconds = 0;
21:     char strLogMessage[512];
22:     while (1)
23:     {
24:         iFileFd = OpenLogFile();
25:
26:         SDM_GetTime(&uiSeconds, &uiUSeconds);
27:
28:         snprintf(strLogMessage, sizeof(strLogMessage),
29:             "SDM_GetTime returned %u seconds %u useconds. \n",
30:             uiSeconds, uiUSeconds);
31:
32:         if (write (iFileFd, strLogMessage, strlen(strLogMessage)) < 0)
33:         {
34:             perror("write:");
35:             printf("%d \n", iFileFd);
36:             return -1;
37:         }
38:
39:         close(iFileFd);
```

```
40:     sleep(1);
41: }
42: return 0;
43: }
44:
45: int OpenLogFile()
46: {
47:     int iResult = 0;
48:     if ((iResult = open (g_strLogFileName, O_RDWR | O_TRUNC | O_CREAT, S_IRUSR |
S_IWUSR)) < 0)
49:     {
50:         perror("open:");
51:         exit(EXIT_FAILURE);
52:     }
53:     return iResult;
54: }
55:
56:
```

File: sdm/app/test/TaskPostTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5: #include "../common/message/SDMCommand.h"
6: #include "../common/message/SDMReqReg.h"
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/MessageManager/MessageManager.h"
9: #include "../common/MessageManipulator/MessageManipulator.h"
10: #include "../common/task/taskdefs.h"
11: #include "../tm/tm.h"
12:
13:
14: void RequestTmMessages();
15: void RegInfoHandler(void*);
16: int StartTaskMenu();
17:
18: enum ReqRegIds { ID_START_TASK };
19:
20: long g_lPort = 0;
21:
22: //
23: // TM Data
24: SDMComponent_ID g_cidTaskManager;
25: // StartTask data
26: SDMMMessage_ID g_midStartTask;
27: MessageManipulator g_mmStartTask;
28: bool g_bStartTaskFound = false;
29:
30:
31: int main (int argc, char** argv)
32: {
33:     char msgBuf[BUFSIZE];
34:     MessageManager mm;
35:
36:     SDMInit(argc, argv);
37:
38:     if ((g_lPort = getPort()) == SDM_PM_NOT_AVAILABLE)
39:     {
```



```

40:     g_lPort = 7354; //Arbitrary
41: }
42:
43: mm.Async_Init(g_lPort);
44:
45: printf("Finding TM messages. \n");
46: RequestTmMessages();
47:
48: char cMessageType;
49: while (false == g_bStartTaskFound)
50: {
51:     if (mm.IsReady())
52: {
53: cMessageType = mm.GetMessage(msgBuf);
54: switch(cMessageType)
55: {
56:     case SDM_RegInfo:
57:         RegInfoHandler(msgBuf);
58:         break;
59:     default:
60:         printf("Unexpected message 0x%hhx (%c) \n", cMessageType, cMessageType);
61:         break;
62:     }
63: }
64:     else
65: {
66:     sleep (1);
67: }
68: }
69:
70: while (0 != StartTaskMenu())
71: ;
72:
73: return 0;
74: }
75:
76: int StartTaskMenu()
77: {
78: char strTaskName[128];
79: printf ("Enter the name of the task to start (exit to exit): ");
80: scanf ("%s", strTaskName);

```

```

81:
82: if (0 == strcmp(strTaskName, "exit"))
83:     return 0;
84:
85: char cAnswer;
86: unsigned short usResources = SDM_INTEL | SDM_LINUX26 | SDM_MEM128 | PM_ID(3);
87: int iMode = 1;
88: char cArchType, cOsType, cMemType, cExecutionMode, cLoadLocation;
89:
90:
91: printf ("Enter the Architecture type. \n");
92: printf ("1) ARCH_TYPE_INTEL \n");
93: printf ("2) ARCH_TYPE_MICROBLAZE \n");
94: printf (" : ");
95: scanf ("%hhd", &cAnswer);
96: switch (cAnswer)
97: {
98:     case 1:
99:         cArchType = ARCH_TYPE_INTEL;
100:         break;
101:     case 2:
102:         cArchType = ARCH_TYPE_MICROBLAZE;
103:         break;
104: }
105:
106: printf("Enter the Operating system type. \n");
107: printf("1) OS_TYPE_LINUX26 \n");
108: printf("2) OS_TYPE_WIN32 \n");
109: printf(" : ");
110: scanf ("%hhd", &cAnswer);
111: switch (cAnswer)
112: {
113:     case 1:
114:         cOsType = OS_TYPE_LINUX26;
115:         break;
116:     case 2:
117:         cOsType = OS_TYPE_WIN32;
118:         break;
119: }
120:
121: printf("Enter the memory type. \n");

```

```

122: printf("1) MEM_TYPE_64 \n");
123: printf("2) MEM_TYPE_128 \n");
124: printf("3) MEM_TYPE_256 \n");
125: printf("4) MEM_TYPE_512 \n");
126: usResources |= SDM_MEM128;
127: printf(" : ");
128: scanf ("%hhhd", &cAnswer);
129: switch (cAnswer)
130: {
131:     case 1:
132:         cMemType = MEM_TYPE_64;
133:         break;
134:     case 2:
135:         cMemType = MEM_TYPE_128;
136:         break;
137:     case 3:
138:         cMemType = MEM_TYPE_256;
139:         break;
140:     case 4:
141:         cMemType = MEM_TYPE_512;
142:         break;
143: }
144:
145: printf("Enter the execution mode. \n");
146: printf("1) EXECUTION_MODE_NORMAL \n");
147: printf("2) EXECUTION_MODE_ALWAYS_RUNNING \n");
148: printf(" : ");
149: scanf ("%hhhd", &cAnswer);
150: switch (cAnswer)
151: {
152:     case 1:
153:         cExecutionMode = EXECUTION_MODE_NORMAL;
154:         break;
155:     case 2:
156:         cExecutionMode = EXECUTION_MODE_ALWAYS_RUNNING;
157:         break;
158: }
159:
160: printf("Enter the task load location. \n");
161: printf("1) TASK_LOCATION_PRIMARY \n");
162: printf("2) TASK_LOCATION_TEMPORARY \n");

```

```

163: printf("3) TASK_LOCATION_BACKUP \n");
164: printf(" : ");
165: scanf("%hhd", &cAnswer);
166: switch (cAnswer)
167: {
168:     case 1:
169:         cLoadLocation = TASK_LOCATION_PRIMARY;
170:         break;
171:     case 2:
172:         cLoadLocation = TASK_LOCATION_TEMPORARY;
173:         break;
174:     case 3:
175:         cLoadLocation = TASK_LOCATION_BACKUP;
176:         break;
177: }
178:
179: unsigned int uiBufferOffset = 0;
180: // Put together the message
181: SDMCommand msgCommand;
182: msgCommand.source = g_cidTaskManager;
183: msgCommand.command_id = g_midStartTask;
184: msgCommand.length = g_mmStartTask.getLength(COMMANDMSG);
185: strcpy(msgCommand.data, strTaskName);
186: uiBufferOffset += XTEDS_MAX_TASK_NAME_SIZE;
187:
188: PUT_CHAR( &msgCommand.data[uiBufferOffset], cArchType);
189: uiBufferOffset += sizeof(char);
190:
191: PUT_CHAR( &msgCommand.data[uiBufferOffset], cOsType);
192: uiBufferOffset += sizeof(char);
193:
194: PUT_CHAR( &msgCommand.data[uiBufferOffset], cMemType);
195: uiBufferOffset += sizeof(char);
196:
197: PUT_CHAR( &msgCommand.data[uiBufferOffset], cExecutionMode);
198: uiBufferOffset += sizeof(char);
199:
200: PUT_CHAR( &msgCommand.data[uiBufferOffset], cLoadLocation);
201: uiBufferOffset += sizeof(char);
202:
203: msgCommand.Send();

```

```

204: return -1;
205: }
206:
207: void RegInfoHandler(void* pMessageBuffer)
208: {
209:     char* pMessage = static_cast<char*>(pMessageBuffer);
210:     SDMRegInfo msgInfo;
211:
212:     long lResult = 0;
213:     if ((lResult = msgInfo.Unmarshal(pMessage)) == SDM_NO_FURTHER_DATA_PROVIDER)
214:     {
215:         return;
216:     }
217:     else if (lResult < 0)
218:     {
219:         printf("Invalid SDMRegInfo message. \n");
220:         return;
221:     }
222:
223:     if (msgInfo.id == ID_START_TASK)
224:     {
225:         if (msgInfo.type == SDM_REGINFO_REGISTRATION)
226:         {
227:             g_mmStartTask.setMsgDef(msgInfo.msg_def);
228:             g_midStartTask = msgInfo.msg_id;
229:             g_cidTaskManager = msgInfo.source;
230:             g_bStartTaskFound = true;
231:         }
232:         else if (msgInfo.type == SDM_REGINFO_CANCELLATION)
233:         {
234:             g_bStartTaskFound = false;
235:         }
236:     }
237: }
238:
239: void RequestTmMessages()
240: {
241:     SDMReqReg msgRequest;
242:     msgRequest.destination.setPort(g_lPort);
243:     strncpy(msgRequest.device, "TaskManager", sizeof(msgRequest.device));
244:     strncpy(msgRequest.item_name, "StartTask", sizeof(msgRequest.item_name));

```

```
245: msgRequest.id = ID_START_TASK;
246: msgRequest.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
247:
248: msgRequest.Send();
249: }
```

File: sdm/app/test/MessageLogAdd.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMCommand.h"
7: #include "../common/message/SDMComponent_ID.h"
8: #include "../common/MessageManipulator/MessageManipulator.h"
9: #include "../common/MessageManager/MessageManager.h"
10:
11: #include <string.h>
12: #include <unistd.h>
13: #include <stdio.h>
14: #include <sys/types.h>
15: #include <sys/wait.h>
16: #include <signal.h>
17: #include <sys/socket.h>
18: #include <netinet/in.h>
19: #include <arpa/inet.h>
20:
21:
22: #define DATA_PROVIDER    1
23:
24: long my_port;
25: SDMComponent_ID data_provider;
26: SDMComponent_ID service_provider;
27: unsigned char service_msg = 0;
28:
29: MessageManipulator data_manipulator;
30:
31: void RegInfoHandler(SDMRegInfo& info)
32: {
33:     SDMCommand command;
34:     SDMReqReg req_reg;
35:     char type;
36:     SDMComponent_ID comp_id;
37:     char ip_address[16];
38:     int sensor_id;
39:     int port;
```

```

40:
41: //Set the port we will be receiving on
42: command.destination.setPort(my_port);
43: //copy the sensor id into the consume message
44: command.source=info.source;
45: //copy the msg id into the consume message
46: command.command_id=info.msg_id;
47:
48: if (info.id != DATA_PROVIDER)
49:     return;
50:
51: if (info.source.getAddress() == TaskManager.getAddress() && info.source.getPort() ==
TaskManager.getPort())
52: {
53:     printf("Send log command to TM? (y/n) ");
54:     type = getchar();
55:     getchar();
56:     if (type == 'n' || type == 'N')
57:         return;
58: }
59: else if (info.source.getAddress() == DataManager.getAddress() && info.source.getPort() ==
DataManager.getPort())
60: {
61:     printf("Send log command to DM? (y/n) ");
62:     type = getchar();
63:     getchar();
64:     if (type == 'n' || type == 'N')
65:         return;
66: }
67: else if (info.source.getPort() == PORT_SM)
68: {
69:     printf("Send log command to SM? (y/n) ");
70:     type = getchar();
71:     getchar();
72:     if (type == 'n' || type == 'N')
73:         return;
74: }
75: else if (info.source.getPort() == PORT_PM)
76: {
77:     printf("Send log command to PM? (y/n) ");
78:     type = getchar();

```



```

79:     getchar();
80:     if (type == 'n' || type == 'N')
81:         return;
82: }
83:
84: printf("Log for a specific component id? (y/n) ");
85: type = getchar();
86: getchar();
87: if (type == 'y' || type == 'Y')
88: {
89:     printf("Enter the IP address (number-dot notation): ");
90:     scanf("%s", ip_address);
91:     getchar();
92:     printf("Enter the port: ");
93:     scanf("%d", &port);
94:     getchar();
95:     printf("Enter the sensor id: ");
96:     scanf("%d", &sensor_id);
97:     getchar();
98:     if (!strcmp(ip_address,"0"))
99:         comp_id.setAddress(0);
100:     else
101:         comp_id.setAddress(inet_addr(ip_address));
102:     comp_id.setPort(port);
103:     comp_id.setSensorID(sensor_id);
104: }
105: else
106: {
107:     comp_id.setAddress(0);
108:     comp_id.setPort(0);
109:     comp_id.setSensorID(0);
110: }
111:
112: data_provider = info.source;
113: data_manipulator.setMsgDef(info.msg_def);
114: command.length = 12;
115: printf(" \n \n%s \n",info.msg_def);
116: printf(" \nEnter the character representation of the message types to log. \n");
117: printf("Example: typing  \tu[ENTER] \n will add both SDMData and SDMSubreqst messages.
\n");
118: printf("--> ");

```

```

119: while ((type = getchar()) != '\n')
120: {
121:     switch (type)
122:     {
123:         case 'a':
124:         case 'b':
125:         case 'c':
126:         case 'd':
127:         case 'e':
128:         case 'f':
129:         case 'g':
130:         case 'h':
131:         case 'i':
132:         case 'j':
133:         case 'k':
134:         case 'l':
135:         case 'm':
136:         case 'n':
137:         case 'o':
138:         case 'p':
139:         case 'q':
140:         case 'r':
141:         case 's':
142:         case 't':
143:         case 'u':
144:         case 'v':
145:         case 'w':
146:         case 'x':
147:         case 'y':
148:         case 'z':
149:         case 'A':
150:         case 'B':
151:         case 'C':
152:         case 'D':
153:         case 'E':
154:         case 'F':
155:         case 'G':
156:         case 'H':
157:         case 'I':
158:         case 'J':
159:             PUT_UCHAR(&command.data[0], type);

```

```

160:         comp_id.Marshal(command.data, 1);
161:         command.Send();
162:         printf("    Command message sent for message type '%c'. \n",type);
163:         break;
164:     case '0':    //Log everything
165:         PUT_UCHAR(&command.data[0], '\0');
166:         comp_id.Marshal(command.data, 1);
167:         command.Send();
168:         printf("    Command message sent to log all messages. \n");
169:         break;
170:     }
171:     usleep(2000);
172: }
173: }
174:
175: int main(int argc,char** argv)
176: {
177:     MessageManager mm;
178:     SDMDData dat;
179:     SDMRegInfo info;
180:     SDMReqReg req_reg;
181:     char buf[BUFSIZE];
182:     long length;
183:     bool infosDone = false;
184:     int numRecd = 0;
185:
186:     //initialize consumer
187:     SDMInit(argc,argv);
188:     my_port = getPort();
189:     mm.Async_Init(my_port);
190:
191:     printf("Consumer listening on port %ld \n",my_port);
192:
193:     while(!infosDone)
194:     {
195:         if (mm.IsReady())
196:         {
197:             switch(mm.GetMessage(buf,length))
198:             {
199:                 case SDM_RegInfo:
200:                     if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)

```

```

201:         {
202:             numRecd++;
203:             RegInfoHandler(info);
204:         }
205:     else
206:     {
207:         if (numRecd == 0)
208:         {
209:             printf("No log messages found. \n");
210:             fflush(NULL);
211:         }
212:         infosDone = true;
213:     }
214:     break;
215: default:
216:     printf("Unexpected message \n");
217: }
218: }
219: else
220: { //check for data and service providers
221:     if(data_provider.getSensorID() == 0)
222:     {
223:         //request info on integer data providers
224:         //Set variable name
225:         strcpy(req_reg.item_name,"Enable_Logging");
226:         //Set the quallist can be empty
227:         strcpy(req_reg.quallist,"< >");
228:         req_reg.reply = SDM_REQREG_CURRENT_AND_FUTURE;
229:         req_reg.destination.setPort(my_port);
230:         req_reg.id = DATA_PROVIDER;
231:         req_reg.Send();
232:         printf("Searching for log command... \n");
233:         sleep(2);
234:     }
235:     usleep(1000);
236: }
237: }
238: }

```

File: sdm/app/test/StressxTEDtest.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMData.h"
3: #include "../common/message/SDMCancelxTEDS.h"
4: #include <string.h>
5: #include <sys/types.h>
6: #include <sys/stat.h>
7: #include <fcntl.h>
8: #include <unistd.h>
9: #include <stdio.h>
10: #include <stdlib.h>
11: #include <pthread.h>
12:
13: void RegisterxTEDS();
14: void CancelxTEDS();
15:
16: long my_port;
17:
18: int main(int argc, char** argv)
19: {
20:     SDMData dat;
21:
22:     SDMInit(argc, argv);
23:     my_port = getPort();
24:     RegisterxTEDS();
25:     CancelxTEDS();
26: }
27:
28: void RegisterxTEDS()
29: {
30:     SDMxTEDS xteds; // create an xTEDS registration message
31:
32:     strcpy (xteds.xTEDS, "<!--Sample XML file generated by XMLSpy v2005 rel. 3 U
(http://www.altova.com)--> \n \n<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<!-- Th-is is a \t
com-ment wit-h n-ame \nin the co-mment --> \n<xTEDS xmlns=
\"http://www.interfacecontrol.com/SPA/xTEDS \" xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-
instance \" <!-- comment --> name= \"ParticleFluxService_xTEDS \" version= \"2.0 \" description=
\"xTEDS For Particle Flux Service \"> \n \n \t<Device name= \"ParticleFluxService \" id= \"11 \" kind=
\"Software \" description= \"Particle Flux Data Processing Service \"> \n \t \t<Qualifier value= \"heat \"
name= \"Head \" units= \"none \"/> \n \t \t<Location x= \"a \" z= \"c \" y= \"b \" units= \"d \"/></Location>
\n \t \t<Orientation axis= \"Y \" angle= \"10 \" units= \"beta \"/> \n \t \t<!-- --><Orientation axis= \"Z
\" angle= \"16 \" units= \"degrees \"/></Orientation> \n \t</Device> \n \n \t<!-- comment --><!-- co-mme-
```

```

nt --><Interface name= \"Particle_Interface \" id= \"1 \"> \n \t \t<Variable name= \"severity \" kind=
\"Severity Level \" format= \"UINT08 \" description= \"the severity level of data \"/> \n \t \t<Variable
name= \"trend \" kind= \"Trend \" format= \"UINT08 \" description= \"The trend of data progression \"/>
\n \t \t<Variable name= \"startTS \" format= \"UINT08 \" kind= \"time_stamp \" length= \"15 \"
description= \"Starting timestamp of data making up this result \"/> \n \t \t<Variable name= \"endTS \"
format= \"UINT08 \" kind= \"time_stamp \" length= \"15 \" description= \"Ending timestamp of data
making up this result \"/> \n \n \t \t<Notification> \n \t \t \t<DataMsg name= \"particleFluxData \"
msgArrival= \"PERIODIC \" msgRate= \"Stream \" id= \"12 \" description= \"Particle Flux data
processing interim result \"> \n \t \t \t \t<VariableRef name= \"severity \"/> \n \t \t \t \t<VariableRef
name= \"trend \"/> \n \t \t \t \t<VariableRef name= \"startTS \"/> \n \t \t \t \t<VariableRef name= \"endTS
\"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t</Interface> \n \n</xTEDS> \n\")); // set xTEDS

```

```
33:
```

```
34: xteds.source.setSensorID(1);           // set the id of this application
```

```
35: xteds.source.setPort(my_port);
```

```
36: printf("Registering producer xTEDS on port %ld \n",my_port);
```

```
37: xteds.Send();                         // register with the SDM
```

```
38: /*xteds.Send();                      // register with the SDM
```

```
39: xteds.Send();                         // register with the SDM
```

```
40: xteds.Send();                         // register with the SDM
```

```
41: xteds.Send();                         // register with the SDM
```

```
42: xteds.Send();                         // register with the SDM
```

```
43: xteds.Send();                         // register with the SDM
```

```
44: xteds.Send();                         // register with the SDM
```

```
45: xteds.Send();                         // register with the SDM
```

```
46: xteds.Send();                         // register with the SDM*/
```

```
47: }
```

```
48:
```

```
49: void CancelxTEDS()
```

```
50: {
```

```
51: SDMCancelxTEDS cancel;
```

```
52: printf("Canceling xTEDS \n");
```

```
53: cancel.source.setSensorID(1);
```

```
54: cancel.source.setPort(my_port);
```

```
55: cancel.Send();
```

```
56: }
```

File: sdm/app/test/FaultMsgtest.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMData.h"
3: #include "../common/message/SDMCancelxTEDS.h"
4: #include "../common/message/SDMReqReg.h"
5: #include <string.h>
6: #include <sys/types.h>
7: #include <sys/stat.h>
8: #include <fcntl.h>
9: #include <unistd.h>
10: #include <stdio.h>
11: #include <stdlib.h>
12: #include <pthread.h>
13:
14: void RegisterxTEDS();
15: void CancelxTEDS();
16: void RequestRegistration();
17:
18: long my_port;
19:
20: int main(int argc, char** argv)
21: {
22:     SDMData dat;
23:
24:     SDMInit(argc, argv);
25:     my_port = getPort();
26:     RegisterxTEDS();
27:     RequestRegistration();
28:     CancelxTEDS();
29: }
30:
31: void RegisterxTEDS()
32: {
33:     SDMxTEDS xteds; // create an xTEDS registration message
34:
35:     strcpy (xteds.xTEDS, "<!--Sample XML file generated by XMLSpy v2005 rel. 3 U
(http://www.altova.com)--> \n \n<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<!-- Th-is is a \t
com-ment wit-h n-ame \nin the co-mment --> \n<xTEDS xmlns=
\"http://www.interfacecontrol.com/SPA/xTEDS \" xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-
instance \" <!-- comment --> name= \"ParticleFluxService_xTEDS \" version= \"2.0 \" description=
\"xTEDS For Particle Flux Service \"> \n \n \t<Device name= \"ParticleFluxService \" id= \"11 \" kind=
```

```

"Software \" description= \"Particle Flux Data Processing Service \"/> \n \t \t<Qualifier value= \"heat \"
name= \"Head \" units= \"none \"/> \n \t \t<Location x= \"a \" z= \"c \" y= \"b \" units= \"d \"/></Location>
\n \t \t<Orientation axis= \"Y \" angle= \"10 \" units= \"beta \"/> \n \t \t<!--      --><Orientation axis= \"Z
\" angle= \"16 \" units= \"degrees \"/></Orientation> \n \t</Device> \n \n \t<!-- comment --><!-- co-mme-
nt --><Interface name= \"Particle_Interface \" id= \"1 \"/> \n \t \t<Variable name= \"severity \" kind=
\"Severity Level \" format= \"UINT08 \" description= \"the severity level of data \"/> \n \t \t<Variable
name= \"trend \" kind= \"Trend \" format= \"UINT08 \" description= \"The trend of data progression \"/>
\n \t \t<Variable name= \"startTS \" format= \"UINT08 \" kind= \"time_stamp \" length= \"15 \"
description= \"Starting timestamp of data making up this result \"/> \n \t \t<Variable name= \"endTS \"
format= \"UINT08 \" kind= \"time_stamp \" length= \"15 \" description= \"Ending timestamp of data
making up this result \"/> \n \n \t \t<Notification> \n \t \t \t<DataMsg name= \"particleFluxData \"
msgArrival= \"PERIODIC \" msgRate= \"Stream \" id= \"12 \" description= \"Particle Flux data
processing interim result \"/> \n \t \t \t \t<VariableRef name= \"severity \"/> \n \t \t \t \t<VariableRef
name= \"trend \"/> \n \t \t \t \t<VariableRef name= \"startTS \"/> \n \t \t \t \t<VariableRef name= \"endTS
\"/> \n \t \t \t</DataMsg> \n \t \t \t<FaultMsg name= \"fault \" id= \"13 \"/> \n \t \t</Notification> \n
\t</Interface> \n \n</xTEDS> \n"); // set xTEDS

```

36:

```
37: xteds.source.setSensorID(1);          // set the id of this application
```

```
38: xteds.source.setPort(my_port);
```

```
39: printf("Registering producer xTEDS on port %ld \n",my_port);
```

```
40: xteds.Send();                        // register with the SDM
```

```
41: }
```

42:

```
43: void CancelxTEDS()
```

```
44: {
```

```
45: SDMCancelxTEDS cancel;
```

```
46: printf("Canceling xTEDS \n");
```

```
47: cancel.source.setSensorID(1);
```

```
48: cancel.source.setPort(my_port);
```

```
49: cancel.Send();
```

```
50: }
```

51:

```
52: void RequestRegistration()
```

```
53: {
```

```
54: SDMReqReg req;
```

```
55:
```

```
56: //req.source.setSensorID(0x10001);
```

```
57: req.reply = SDM_REQREG_CURRENT;
```

```
58: strcpy(req.item_name,"fault");
```

```
59: printf("Sending ReqReg \n");
```

```
60: req.Send();
```

```
61:
```

```
62: }
```


File: sdm/app/test/badxTEDS1.cpp

```
1: #include "../common/message/SDMmessage.h"
2: #include "../common/message/SDMxTEDS.h"
3: #include "../common/message/SDMCancelxTEDS.h"
4: #include "../common/UDPcom.h"
5: #include "../common/message_defs.h"
6:
7: #include <stdio.h>
8: #include <unistd.h>
9: #include <stdlib.h>
10: #include <string.h>
11: #include <sys/socket.h>
12: #include <netinet/in.h>
13: #include <arpa/inet.h>
14:
15: void nullxteds_withclass();
16: void nullxteds_raw();
17: void nullxteds_incomplete();
18: void garbagexteds();
19: void repeatingxteds();
20: void invalidxteds();
21:
22: int main(int argc, char** argv)
23: {
24:     SDMInit(argc,argv);
25:     nullxteds_withclass();
26:     nullxteds_raw();
27:     nullxteds_incomplete();
28:     garbagexteds();
29:     repeatingxteds();
30:     invalidxteds();
31: }
32:
33: //Sends a NULL xTEDS using the SDMxTEDS class
34: void nullxteds_withclass()
35: {
36:     SDMxTEDS xted;
37:     xted.source.setSensorID(1);
38:     xted.source.setPort(4000);
39:     memset(xted.xTEDS,0,10);
```

```

40: printf("Starting null xTEDS (with class) test \n");
41: xted.Send();
42: sleep(5);
43: puts("Test Done");
44: }
45:
46: //Sends a NULL xTEDS avoiding the SDMxTEDS class
47: void nullxteds_raw()
48: {
49: char msg[BUFSIZE];
50: char geo_location;
51: long id;
52:
53: memset(msg,0,BUFSIZE);
54: id = 2;
55: msg[0] = SDM_xTEDS;
56: memcpy(msg+1,&id,4);
57: id = 4001;
58: memcpy(msg+5,&id,4);
59: geo_location = 0;
60: memcpy(msg+9,&geo_location,1);
61: memcpy(msg+10,&id,4);
62: printf("Starting null xTEDS (raw) test \n");
63: struct in_addr inaddr;
64: inaddr.s_addr = DataManager.getAddress();
65: UDPsendto(inet_ntoa(inaddr),PORT_DM,msg,BUFSIZE);
66: sleep(5);
67: puts("Test Done");
68: }
69:
70: //Inserts a null character into a valid xTEDS at a random location then sends it
71: void nullxteds_incomplete()
72: {
73: SDMxTEDS xted;
74: xted.source.setSensorID(1);
75: xted.source.setPort(4000);
76: int pos;
77: char* xted_text = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS version= \"2.0 \"
name= \"Task_Manager_xTEDS2 \"> \n \t<Application kind= \"Software \" name= \"TaskManager2 \"/>
\n \t<Interface name= \"TM_Interface \" id= \"1 \"> \n \t \t<Variable format= \"UINT08 \" kind= \"TBD \"
name= \"Mode \"/> \n \t \t<Variable format= \"UINT08 \" kind= \"TBD \" name= \"TaskName \" length=
'16 \"/> \n \t \t<Variable format= \"INT32 \" kind= \"TBD \" name= \"ExitStatus \"/> \n \t

```

```

<Notification> \n \t \t <DataMsg name= \"Status \" id= \"1 \" msgArrival= \"EVENT \">> \n \t \t \t
<VariableRef name= \"Mode \"/> \n \t \t </DataMsg> \n \t </Notification> \n \t <Notification> \n \t
\t <DataMsg name= \"TaskQueued \" id= \"2 \" msgArrival= \"EVENT \">> \n \t \t \t <VariableRef
name= \"TaskName \"/> \n \t \t </DataMsg> \n \t </Notification> \n \t <Notification> \n \t \t
<DataMsg name= \"TaskStarted \" id= \"3 \" msgArrival= \"EVENT \">> \n \t \t \t <VariableRef name=
\"TaskName \"/> \n \t \t </DataMsg> \n \t </Notification> \n \t <Notification> \n \t \t <DataMsg
name= \"TaskFinished \" id= \"4 \" msgArrival= \"EVENT \">> \n \t \t \t <VariableRef name=
\"TaskName \"/> \n \t \t \t <VariableRef name= \"ExitStatus \"/> \n \t \t </DataMsg> \n \t
</Notification> \n \t <Command> \n \t \t <CommandMsg name= \"ChangeMode \" id= \"5 \">> \n \t \t
\t <VariableRef name= \"Mode \"/> \n \t \t </CommandMsg> \n \t </Command> \n </Interface> \n
\t <Interface name= \"Msg_Count \" id= \"2 \">> \n \t <Variable name= \"Total_Messages_Recd \" kind=
\"Total \" format= \"UINT16 \"/> \n \t <Variable name= \"Messages_Last_Second_Recd \" kind=
\"Total \" format= \"UINT16 \"/> \n \t <Variable name= \"Total_Messages_Sent \" kind= \"Total \"
format= \"UINT16 \"/> \n \t <Variable name= \"Messages_Last_Second_Sent \" kind= \"Total \"
format= \"UINT16 \"/> \n \n \t <Notification> \n \t \t <DataMsg name= \"Message_Count \" id= \"13 \"
msgArrival= \"PERIODIC \">> \n \t \t \t <VariableRef name= \"Total_Messages_Recd \"/> \n \t \t \t
\t <VariableRef name= \"Messages_Last_Second_Recd \"/> \n \t \t \t \t <VariableRef name=
\"Total_Messages_Sent \"/> \n \t \t \t \t <VariableRef name= \"Messages_Last_Second_Sent \"/> \n \t \t
\t </DataMsg> \n \t </Notification> \n \t </Interface> \n \t <Interface name= \"Message_Log \" id= \"3 \">>
\t \t <Variable format= \"UINT08 \" kind= \"TBD \" name= \"Msg_Type \"/> \n \t \t <Command> \n \t \t
\t <CommandMsg name= \"Enable_Logging \" id= \"16 \">> \n \t \t \t <VariableRef name= \"Msg_Type
\"/> \n \t \t \t </CommandMsg> \n \t \t </Command> \n \t \t <Command> \n \t \t <CommandMsg name=
\"Disable_Logging \" id= \"17 \">> \n \t \t \t \t <VariableRef name= \"Msg_Type \"/> \n \t \t
\t </CommandMsg> \n \t \t </Command> \n </Interface> \n </xTEDS>;

```

78:

79: printf("Starting null xTEDS (incomplete) test \n");

80: for(int i=0;i<5;i++)

81: {

82: printf("%d.",i);

83: pos = (rand() % 300) + 10;

84: strcpy(xted.xTEDS,xted_text);

85: xted.xTEDS[pos] = '\0';

86: xted.Send();

87: }

88: sleep(5);

89: puts(" \nTest Done");

90: }

91:

92: //Sends garbage bytes as an xTEDS

93: void garbagexteds()

94: {

95: SDMxTEDS xted;

96: xted.source.setSensorID(1);

97: xted.source.setPort(4000);

98:

```

99: printf("Starting garbage xTEDS test \n");
100:  for(int i=0;i<5;i++)
101:  {
102:      for(int j=0;j<500;j++)
103:          xted.xTEDS[j] = (char)((rand() % 255) + 1);
104:      printf("%d.",i);
105:      xted.Send();
106:  }
107:  sleep(5);
108:  puts(" \nTest Done");
109: }
110:
111: //Sends multiple xTEDS in an effort to fill the DM's xTED buffer
112: void repeatingxteds()
113: {
114:     SDMxTEDS xted;
115:     SDMCancelxTEDS cancel;
116:     cancel.source.setSensorID(1);
117:     cancel.source.setPort(4000);
118:     xted.source.setSensorID(1);
119:     xted.source.setPort(4000);
120:     char* xted_text = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS version= \"2.0 \"
name= \"Task_Manager_xTEDS \"> \n \t<Application kind= \"Software \" name= \"TaskManager \"/> \n
\t<Interface name= \"TM_Interface \" id= \"1 \"> \n \t \t<Variable format= \"UINT08 \" kind= \"TBD \"
name= \"Mode \"/> \n \t \t<Variable format= \"UINT08 \" kind= \"TBD \" name= \"TaskName \" length=
\"16 \"/> \n \t \t \t<Variable format= \"INT32 \" kind= \"TBD \" name= \"ExitStatus \"/> \n \t
\t<Notification> \n \t \t \t<DataMsg name= \"Status \" id= \"1 \" msgArrival= \"EVENT \"> \n \t \t \t
\t<VariableRef name= \"Mode \"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t \t<Notification> \n \t
\t \t<DataMsg name= \"TaskQueued \" id= \"2 \" msgArrival= \"EVENT \"> \n \t \t \t \t<VariableRef
name= \"TaskName \"/> \n \t \t \t \t</DataMsg> \n \t \t</Notification> \n \t \t<Notification> \n \t \t
\t<DataMsg name= \"TaskStarted \" id= \"3 \" msgArrival= \"EVENT \"> \n \t \t \t \t<VariableRef name=
\"TaskName \"/> \n \t \t \t \t</DataMsg> \n \t \t</Notification> \n \t \t<Notification> \n \t \t \t<DataMsg
name= \"TaskFinished \" id= \"4 \" msgArrival= \"EVENT \"> \n \t \t \t \t \t<VariableRef name=
\"TaskName \"/> \n \t \t \t \t \t<VariableRef name= \"ExitStatus \"/> \n \t \t \t \t</DataMsg> \n \t
\t</Notification> \n \t \t<Command> \n \t \t \t<CommandMsg name= \"ChangeMode \" id= \"5 \"> \n \t \t
\t \t<VariableRef name= \"Mode \"/> \n \t \t \t \t</CommandMsg> \n \t \t</Command> \n \t</Interface> \n
\t<Interface name= \"Msg_Count \" id= \"2 \"> \n \t \t<Variable name= \"Total_Messages_Recd \" kind=
\"Total \" format= \"UINT16 \"/> \n \t \t \t<Variable name= \"Messages_Last_Second_Recd \" kind=
\"Total \" format= \"UINT16 \"/> \n \t \t \t<Variable name= \"Total_Messages_Sent \" kind= \"Total \"
format= \"UINT16 \"/> \n \t \t \t \t<Variable name= \"Messages_Last_Second_Sent \" kind= \"Total \"
format= \"UINT16 \"/> \n \t \t \t \t<Notification> \n \t \t \t \t<DataMsg name= \"Message_Count \" id= \"13 \"
msgArrival= \"PERIODIC \"> \n \t \t \t \t \t<VariableRef name= \"Total_Messages_Recd \"/> \n \t \t \t \t
\t<VariableRef name= \"Messages_Last_Second_Recd \"/> \n \t \t \t \t \t \t<VariableRef name=
\"Total_Messages_Sent \"/> \n \t \t \t \t \t \t<VariableRef name= \"Messages_Last_Second_Sent \"/> \n \t \t
\t \t</DataMsg> \n \t \t</Notification> \n \t</Interface> \n \t<Interface name= \"Message_Log \" id= \"3 \">
\n \t \t \t<Variable format= \"UINT08 \" kind= \"TBD \" name= \"Msg_Type \"/> \n \t \t<Command> \n \t \t

```

```

\t<CommandMsg name= \"Enable_Logging \" id= \"16 \"> \n \t \t \t<VariableRef name= \"Msg_Type
\"/> \n \t \t \t</CommandMsg> \n \t \t</Command> \n \t \t<Command> \n \t \t<CommandMsg name=
\"Disable_Logging \" id= \"17 \"> \n \t \t \t \t<VariableRef name= \"Msg_Type \"/> \n \t \t
\t</CommandMsg> \n \t \t</Command> \n \t</Interface> \n</xTEDS>";
121:
122:   strcpy(xted.xTEDS,xted_text);
123:
124:   printf("Starting repeating xTEDS test \n");
125:   for(int i=0;i<20;i++)
126:   {
127:       printf("%d.",i);
128:       xted.Send();
129:   }
130:   sleep(5);
131:   cancel.Send();
132:   puts(" \nTest Done");
133: }
134: void invalidxteds()
135: {
136:   SDMxTEDS xted;
137:   SDMCcancelxTEDS cancel;
138:   cancel.source.setSensorID(1);
139:   cancel.source.setPort(4004);
140:   xted.source.setSensorID(1);
141:   xted.source.setPort(4004);
142:   //Invalid xTEDS "<VariaeRef" should be "<VariableRef"
143:   char* xted_text = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS version= \"2.0 \"
name= \"Task_Manager_xTEDS2 \"> \n \t<Application kind= \"Software \" name= \"TaskManager2 \"/>
\n \t<Interface name= \"TM_Interface \" id= \"1 \"> \n \t \t<Variable format= \"UINT08 \" kind= \"TBD \"
name= \"Mode \"/> \n \t \t<Variable format= \"UINT08 \" kind= \"TBD \" name= \"TaskName \" length=
\"16 \"/> \n \t \t \t<Variable format= \"INT32 \" kind= \"TBD \" name= \"ExitStatus \"/> \n \t
\t<Notification> \n \t \t \t<DataMsg name= \"Status \" id= \"1 \" msgArrival= \"EVENT \"> \n \t \t \t
\t<VariaeRef name= \"Mode \"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t \t<Notification> \n \t \t
\t<DataMsg name= \"TaskQueued \" id= \"2 \" msgArrival= \"EVENT \"> \n \t \t \t \t<VariableRef name=
\"TaskName \"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t \t<Notification> \n \t \t \t<DataMsg
name= \"TaskStarted \" id= \"3 \" msgArrival= \"EVENT \"> \n \t \t \t \t<VariableRef name= \"TaskName
\"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t \t<Notification> \n \t \t \t<DataMsg name=
\"TaskFinished \" id= \"4 \" msgArrival= \"EVENT \"> \n \t \t \t \t<VariableRef name= \"TaskName \"/>
\n \t \t \t \t<VariableRef name= \"ExitStatus \"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t
\t<Command> \n \t \t \t<CommandMsg name= \"ChangeMode \" id= \"5 \"> \n \t \t \t \t<VariableRef
name= \"Mode \"/> \n \t \t \t</CommandMsg> \n \t \t</Command> \n \t</Interface> \n \t<Interface name=
\"Msg_Count \" id= \"2 \"> \n \t \t<Variable name= \"Total_Messages_Recd \" kind= \"Total \" format=
\"UINT16 \"/> \n \t \t<Variable name= \"Messages_Last_Second_Recd \" kind= \"Total \" format=
\"UINT16 \"/> \n \t \t<Variable name= \"Total_Messages_Sent \" kind= \"Total \" format= \"UINT16 \"/>
\n \t \t<Variable name= \"Messages_Last_Second_Sent \" kind= \"Total \" format= \"UINT16 \"/> \n \n \t

```

```

\t<Notification> \n \t \t \t<DataMsg name= \"Message_Count \" id= \"13 \" msgArrival= \"PERIODIC \">
\n \t \t \t \t \t<VariableRef name= \"Total_Messages_Recd \"/> \n \t \t \t \t \t<VariableRef name=
\"Messages_Last_Second_Recd \"/> \n \t \t \t \t \t<VariableRef name= \"Total_Messages_Sent \"/> \n \t \t \t
\t<VariableRef name= \"Messages_Last_Second_Sent \"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n
\t</Interface> \n \t<Interface name= \"Message_Log \" id= \"3 \"> \n \t \t<Variable format= \"UINT08 \"
kind= \"TBD \" name= \"Msg_Type \"/> \n \t \t \t<Command> \n \t \t \t \t<CommandMsg name=
\"Enable_Logging \" id= \"16 \"> \n \t \t \t \t \t<VariableRef name= \"Msg_Type \"/> \n \t \t
\t</CommandMsg> \n \t \t \t</Command> \n \t \t \t<Command> \n \t \t \t \t<CommandMsg name=
\"Disable_Logging \" id= \"17 \"> \n \t \t \t \t \t<VariableRef name= \"Msg_Type \"/> \n \t \t
\t</CommandMsg> \n \t \t</Command> \n \t</Interface> \n</xTEDS>";

```

144:

145: strcpy(xted.xTEDS,xted_text);

146: printf("Invalid xTEDS test. \n");

147: xted.Send();

148: sleep(5);

149: cancel.Send();

150: puts(" \nTest Done");

151: }

152:

153:

File: sdm/app/test/BreadBoardTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <signal.h>
4: #include <unistd.h>
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMCommand.h"
7: #include "../common/message/SDMRegInfo.h"
8: #include "../common/MessageManager/MessageManager.h"
9: #include "../common/message/SDMData.h"
10: #include "../common/message/SDMConsume.h"
11: #include "../common/message/SDMCancel.h"
12:
13: #define BB_SETRATE 1
14: #define BB_VOLTREAD 2
15: #define BB_LCD 3
16:
17: SDMComponent_ID sub_compid;
18: SDMMessage_ID sub_msgid;
19: SDMMessage_ID lcd_msgid;
20:
21: void SigHandler(int signum);
22:
23: int main (int argc, char ** argv)
24: {
25:     MessageManager mm;
26:     SDMReqReg reqreg_msg;
27:     char buf[BUFSIZE];
28:     SDMRegInfo reginfo_msg;
29:     SDMCommand cmd_msg;
30:     SDMData data_msg;
31:     SDMConsume cons_msg;
32:     unsigned char type;
33:     SDMInit(argc, argv);
34:     signal(SIGINT, SigHandler);
35:
36:     mm.Async_Init(4050);
37:
38:     printf(" \nSending ReqReg for breadboard data rate... \n");
39:     strcpy(reqreg_msg.device, "Bread Board");
```

```

40: strcpy(reqreg_msg.interface,"1");
41: strcpy(reqreg_msg.item_name,"Set_Data_Rate");
42: reqreg_msg.id = BB_SETRATE;
43: reqreg_msg.destination.setPort(4050);
44: reqreg_msg.Send();
45:
46: printf("Sending ReqReg for breadboard voltage reading... \n");
47: strcpy(reqreg_msg.item_name,"Voltage_Reading");
48: reqreg_msg.id = BB_VOLTREAD;
49: reqreg_msg.Send();
50:
51: printf("Sending ReqReg for breadboard LCD command... \n");
52: strcpy(reqreg_msg.item_name,"Set_LCD");
53: reqreg_msg.id = BB_LCD;
54: reqreg_msg.Send();
55:
56: unsigned short data = 4;
57: while (1)
58: {
59:     type = mm.BlockGetMessage(buf);
60:     switch (type)
61:     {
62:         case SDM_RegInfo:
63:             if (reginfo_msg.Unmarshal(buf) != SDM_NO_FURTHER_DATA_PROVIDER)
64:             {
65:                 if (reginfo_msg.id == BB_SETRATE)
66:                 {
67:                     cmd_msg.source = reginfo_msg.source;
68:                     cmd_msg.command_id = reginfo_msg.msg_id;
69:                     PUT_USHORT(cmd_msg.data,data);
70:                     cmd_msg.length = 2;
71:                     printf("Sending command message to set breadboard data rate to %hhd. \n",data);
72:                     //cmd_msg.Send(reginfo_msg.source);
73:                     cmd_msg.Send();
74:                 }
75:                 else if (reginfo_msg.id == BB_VOLTREAD)
76:                 {
77:                     cons_msg.source = reginfo_msg.source;
78:                     sub_comp_id = reginfo_msg.source;
79:                     cons_msg.destination.setPort(4050);
80:                     cons_msg.msg_id = reginfo_msg.msg_id;

```



```

81:         sub_msgid = reginfo_msg.msg_id;
82:         printf("Sending consume for breadboard voltage reading... \n");
83:         cons_msg.Send();
84:     }
85:     else if (reginfo_msg.id == BB_LCD)
86:     {
87:         //Save the message ID number
88:         lcd_msgid = reginfo_msg.msg_id;
89:         printf("Saving lcd command message id... \n");
90:     }
91: }
92: break;
93: case SDM_Data:
94: {
95:     unsigned short voltage;
96:     data_msg.Unmarshal(buf);
97:     voltage = GET_USHORT(data_msg.msg+8);
98:     printf("Received voltage reading of %hhd \n",voltage);
99:     //Set the LCD to be the voltage reading
100:     SDMCommand cmd_msg;
101:     cmd_msg.command_id = lcd_msgid;
102:     cmd_msg.source = sub_compid;
103:     PUT_USHORT(cmd_msg.data,voltage);
104:     cmd_msg.length = 2;
105:     cmd_msg.Send();
106:     break;
107: }
108: default:
109:     printf("Unexpected message (%d). \n",type);
110: }
111: }
112:
113:
114: }
115:
116: void SigHandler(int signum)
117: {
118:     SDMCancel cancel_msg;
119:     cancel_msg.source = sub_compid;
120:     cancel_msg.msg_id = sub_msgid;
121:     cancel_msg.destination.setPort(4050);

```

```
122:    cancel_msg.Send();
123:    _exit(0);
124: }
```

File: sdm/app/test/MessageLogRemove.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMCommand.h"
7: #include "../common/MessageManipulator/MessageManipulator.h"
8: #include "../common/MessageManager/MessageManager.h"
9: #include "../common/message/SDMComponent_ID.h"
10:
11: #include <string.h>
12: #include <unistd.h>
13: #include <stdio.h>
14: #include <sys/types.h>
15: #include <sys/wait.h>
16: #include <signal.h>
17: #include <sys/socket.h>
18: #include <netinet/in.h>
19: #include <arpa/inet.h>
20:
21: #define DATA_PROVIDER    1
22:
23: long my_port;
24: SDMComponent_ID data_provider;
25: SDMComponent_ID service_provider;
26: unsigned char service_msg = 0;
27:
28: MessageManipulator data_manipulator;
29:
30: void RegInfoHandler(SDMRegInfo& info)
31: {
32:     SDMCommand command;
33:     SDMReqReg req_reg;
34:     char type;
35:     SDMComponent_ID comp_id;
36:     char ip_address[16];
37:     int sensor_id;
38:     int port;
39:
```

```

40: //Set the port we will be receiving on
41: command.destination.setPort(my_port);
42: //copy the sensor id into the consume message
43: command.source=info.source;
44: //copy the msg id into the consume message
45: command.command_id=info.msg_id;
46: if (info.id != DATA_PROVIDER)
47:     return;
48:
49: if (info.source.getAddress() == TaskManager.getAddress() && info.source.getPort() ==
TaskManager.getPort())
50: {
51:     printf("Send log command to TM? (y/n) ");
52:     type = getchar();
53:     getchar();
54:     if (type == 'n' || type == 'N')
55:         return;
56: }
57: else if (info.source.getAddress() == DataManager.getAddress() && info.source.getPort() ==
DataManager.getPort())
58: {
59:     printf("Send log command to DM? (y/n) ");
60:     type = getchar();
61:     getchar();
62:     if (type == 'n' || type == 'N')
63:         return;
64: }
65: else if (info.source.getPort() == PORT_SM)
66: {
67:     printf("Send log command to SM? (y/n) ");
68:     type = getchar();
69:     getchar();
70:     if (type == 'n' || type == 'N')
71:         return;
72: }
73: else if (info.source.getPort() == PORT_PM)
74: {
75:     printf("Send log command to PM? (y/n) ");
76:     type = getchar();
77:     getchar();
78:     if (type == 'n' || type == 'N')

```

```

79:         return;
80: }
81:
82: printf("Log for a specific component id? (y/n) ");
83: type = getchar();
84: getchar();
85: if (type == 'y' || type == 'Y')
86: {
87:     printf("Enter the IP address (number-dot notation): ");
88:     scanf("%s", ip_address);
89:     getchar();
90:     printf("Enter the port: ");
91:     scanf("%d", &port);
92:     getchar();
93:     printf("Enter the sensor id: ");
94:     scanf("%d", &sensor_id);
95:     getchar();
96:     if (!strcmp(ip_address,"0"))
97:         comp_id.setAddress(0);
98:     else
99:         comp_id.setAddress(inet_addr(ip_address));
100:     comp_id.setPort(port);
101:     comp_id.setSensorID(sensor_id);
102: }
103: else
104: {
105:     comp_id.setAddress(0);
106:     comp_id.setPort(0);
107:     comp_id.setSensorID(0);
108: }
109:
110: data_provider = info.source;
111: data_manipulator.setMsgDef(info.msg_def);
112: command.length = 12;
113: printf(" \n \n%s \n",info.msg_def);
114: printf(" \nEnter the character representation of the message types to remove from logging. \n");
115: printf("Example: typing  \"tu[ENTER] \" will remove both SDMData and SDMSubreqst
messages. \n");
116: printf("--> ");
117: while ((type = getchar()) != '\n')
118: {

```

```

119:     switch (type)
120:     {
121:         case 'a':
122:         case 'b':
123:         case 'c':
124:         case 'd':
125:         case 'e':
126:         case 'f':
127:         case 'g':
128:         case 'h':
129:         case 'i':
130:         case 'j':
131:         case 'k':
132:         case 'l':
133:         case 'm':
134:         case 'n':
135:         case 'o':
136:         case 'p':
137:         case 'q':
138:         case 'r':
139:         case 's':
140:         case 't':
141:         case 'u':
142:         case 'v':
143:         case 'w':
144:         case 'x':
145:         case 'y':
146:         case 'z':
147:         case 'A':
148:         case 'B':
149:         case 'C':
150:         case 'D':
151:         case 'E':
152:         case 'F':
153:         case 'G':
154:         case 'H':
155:         case 'I':
156:         case 'J':
157:             PUT_UCHAR(&command.data[0], type);
158:             comp_id.Marshal(command.data, 1);
159:             command.Send();

```

```

160:         printf("    Command message sent for message type '%c'. \n",type);
161:         break;
162:     case '0':    //Log everything
163:         PUT_UCHAR(&command.data[0], '\0');
164:         comp_id.Marshal(command.data, 1);
165:         command.Send();
166:         printf("    Command message to log all messages. \n");
167:         break;
168:     }
169:     usleep(2000);
170: }
171: }
172:
173: int main(int argc,char** argv)
174: {
175:     MessageManager mm;
176:     SDMDData dat;
177:     SDMRegInfo info;
178:     SDMReqReg req_reg;
179:     char buf[BUFSIZE];
180:     long length;
181:     bool infosDone = false;
182:     int numRecd = 0;
183:
184:     //initialize consumer
185:     SDMInit(argc,argv);
186:     my_port = getPort();
187:     mm.Async_Init(my_port);
188:
189:     printf("Consumer listening on port %ld \n",my_port);
190:
191:     while(!infosDone)
192:     {
193:         if (mm.IsReady())
194:         {
195:             switch(mm.GetMessage(buf,length))
196:             {
197:             case SDM_RegInfo:
198:                 if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)
199:                 {
200:                     numRecd++;

```

```

201:         RegInfoHandler(info);
202:     }
203:     else
204:     {
205:         if (numRecd == 0)
206:         {
207:             printf("No log messages found. \n");
208:             fflush(NULL);
209:         }
210:         infosDone = true;
211:     }
212:     break;
213: default:
214:     printf("Unexpected message \n");
215: }
216: }
217: else
218: { //check for data and service providers
219:     if(data_provider.getSensorID() == 0)
220:     {
221:         //request info on integer data providers
222:         //Set variable name
223:         strcpy(req_reg.item_name,"Disable_Logging");
224:         //Set the quallist can be empty
225:         strcpy(req_reg.quallist,"< >");
226:         req_reg.reply = SDM_REQREG_CURRENT_AND_FUTURE;
227:         req_reg.destination.setPort(my_port);
228:         req_reg.id = DATA_PROVIDER;
229:         req_reg.Send();
230:         printf("Searching for log command... \n");
231:         sleep(2);
232:     }
233:     usleep(1000);
234: }
235: }
236: }

```


File: sdm/app/test/SearchTest.cpp

```
1: #include "../common/message/SDMSearch.h"
2: #include "../common/message/SDMSearchReply.h"
3: #include "../common/MessageManager/MessageManager.h"
4: #include <string.h>
5: #include <sys/types.h>
6: #include <sys/stat.h>
7: #include <fcntl.h>
8: #include <unistd.h>
9: #include <stdio.h>
10: #include <stdlib.h>
11: #include <pthread.h>
12: #include <ctype.h>
13:
14: long my_port;
15:
16: int main(int argc, char** argv)
17: {
18:     SDMSearch s;
19:     SDMSearchReply reply;
20:     int result = 0;
21:     char buf[BUFSIZE];
22:     long length;
23:     MessageManager mm;
24:     int cur = 0;
25:
26:     SDMInit(argc, argv);
27:     my_port = getPort();
28:     mm.Async_Init(my_port);
29:     sleep(1);
30:
31:     s.destination.setPort(my_port);
32:     s.id = 1;
33:
34:     // Return all interface names only
35:     strcpy(s.reg_expr, "<Interface[^>]*name= \"([^\"]*)\" [^>]*>");
36:     s.Send();
37:
38:     // Return the entire <Interface... > tag in the first capture, the interface name
39:     // in the second capture, and the interface id number in the last capture, for
```

```

40: // all interfaces
41: strcpy(s.reg_expr, "<Interface name= \"([^\"]*)\" [^<]*(id= \"[0-9]* \") [^<]*>\"");
42: s.Send();
43:
44: // This regex returns nothing (not a subexpression)
45: strcpy(s.reg_expr, ". *");
46: s.Send();
47:
48: // Returns full xTEDS (is a subexpression)
49: strcpy(s.reg_expr, "(.*)");
50: s.Send();
51:
52: // invalid regular expression (for testing)
53: strcpy(s.reg_expr, "<Interface[^>*name= \"([^\"]*)\" [^>]*>\"");
54: s.Send();
55:
56: while(1)
57: {
58:     if (mm.IsReady())
59:     {
60:         printf(" \n");
61: #ifdef WIN32
62:         mm.GetMsg(buf, length);
63: #else
64:         mm.GetMessage(buf, length);
65: #endif
66:         cur = 0;
67:         result = reply.Unmarshal(buf);
68:         if(result != SDM_NO_FURTHER_DATA_PROVIDER)
69:         {
70:             printf("Match(es)                from                %lu:%lu:%d\n",
reply.source.getSensorID(), reply.source.getAddress(), reply.source.getPort());
71:
72:             bool nullFound = false;
73:             for (unsigned int i = 0; i < sizeof(reply.captured_matches); i++)
74:             {
75:                 if (isprint(reply.captured_matches[i]))
76:                 {
77:                     nullFound = false;
78:                     printf("%c", reply.captured_matches[i]);
79:                 }

```

```

80:         else if (reply.captured_matches[i] == '\0')
81:         {
82:             // Nulls printed as zero
83:             printf("0");
84:             if (nullFound)
85:             {
86:                 printf(" \n");
87:                 break;
88:             }
89:             nullFound = true;
90:         }
91:     }
92: }
93: }
94:     usleep(1000);
95: }
96:
97: return 0;
98: }

```

File: sdm/app/test/VarReq.cpp

```
1: #include "../common/message/SDMmessage.h"
2: #include "../common/message/SDMVarReq.h"
3: #include "../common/UDPcom.h"
4: #include "../common/message_defs.h"
5: #include "../common/MessageManager/MessageManager.h"
6: #include "../common/message/SDMVarInfo.h"
7: #include "../common/message/SDMxTEDS.h"
8: #include "../common/message/SDMCancelxTEDS.h"
9: #include "../common/message/SDMReqReg.h"
10: #include "../common/message/SDMRegInfo.h"
11: #include "../common/message/SDMComponent_ID.h"
12: #include <stdio.h>
13: #include <unistd.h>
14: #include <stdlib.h>
15: #include <string.h>
16: #include <signal.h>
17:
18: void RegisterxTEDS();
19: void CancelxTEDS();
20:
21: long my_port = 0;
22: SDMComponent_ID myId;
23: const int DRANGE = 1;
24: const int CURVE = 2;
25: const int ALL = 3;
26:
27: int main(int argc, char** argv)
28: {
29:     SDMInit(argc,argv);
30:     my_port = getPort();
31:     SDMVarReq request;
32:     SDMVarInfo info;
33:     SDMReqReg reg;
34:     SDMRegInfo reginfo;
35:     MessageManager mm;
36:     char buf[BUFSIZE];
37:     bool found = false;
38:     bool xTEDS = false;
39:     mm.Async_Init(my_port);
```

```

40:
41: RegisterxTEDS();
42: sleep(1);
43: //Get my component id
44: strcpy(reg.interface,"Drange_Interface");
45: reg.source.setPort(my_port);
46: reg.destination.setPort(my_port);
47: reg.Send();
48:
49: while (!found)
50: {
51:     if (mm.IsReady())
52:     {
53:         printf(".");
54:         fflush(NULL);
55:         char type = mm.GetMessage(buf);
56:         switch(type)
57:         {
58:             case SDM_RegInfo:
59:                 int value = reginfo.Unmarshal(buf);
60:                 if (value != SDM_NO_FURTHER_DATA_PROVIDER || value !=
SDM_INVALID_MESSAGE)
61:                 {
62:                     myId = reginfo.source;
63:                     found = true;
64:                     printf("ComponentID found. \n");
65:                 }
66:                 break;
67:             }
68:         }
69:     else
70:     {
71:         sleep(1);
72:     }
73: }
74: CancelxTEDS();
75: request.destination.setPort(my_port);
76: request.id = 1;
77: request.source = myId;
78: request.reply = SDM_VARREQ_CURRENT_AND_FUTURE;
79: printf("Sending VarReq for Drange data. \n");

```

```

80: strcpy(request.variable,"data");
81: request.id = DRANGE;
82: request.msg_id = reginfo.msg_id;
83: request.Send();
84: printf("Sending VarReq for Curve data. \n");
85: strcpy(request.variable,"data");
86: request.id = CURVE;
87: request.msg_id.setInterface(2);
88: request.Send();
89: printf("Sending VarReq for all variables named data in xTEDS \n");
90: request.id = ALL;
91: request.msg_id.setInterface(0);
92: request.Send();
93:
94: found = false;
95: int count = 0;
96: while (!found)
97: {
98:     if (mm.IsReady())
99:     {
100:         char type = mm.GetMessage(buf);
101:         switch(type)
102:         {
103:             case SDM_VarInfo:
104:                 int value = info.Unmarshal(buf);
105:                 if (value != SDM_NO_FURTHER_DATA_PROVIDER && value !=
SDM_INVALID_MESSAGE)
106:                 {
107:                     if (info.id == CURVE)
108:                     {
109:                         printf("Curve variable is %s \n \n",info.var_xTEDS);
110:                         count ++;
111:                     }
112:                     if (info.id == DRANGE)
113:                     {
114:                         printf("Drange variable is %s \n \n",info.var_xTEDS);
115:                         count ++;
116:                     }
117:                     if(info.id == ALL)
118:                     {
119:                         printf("All data variables is %s \n",info.var_xTEDS);

```

```

120:                count++;
121:            }
122:            if (count == 4)
123:                found = true;
124:            fflush(NULL);
125:        }
126:    }
127: }
128: else
129: {
130:     sleep(1);
131:     if(!xTEDS)
132:     {
133:         xTEDS = true;
134:         RegisterxTEDS();
135:     }
136: }
137: }
138: SDMVarReq var;
139: var.reply = SDM_VARREQ_CANCEL;
140: var.destination.setPort(my_port);
141: var.Send();
142: CancelxTEDS();
143: return 1;
144: }
145:
146: void RegisterxTEDS()
147: {
148:     SDMxTEDS xteds; // create an xTEDS registration message
149:
150:     strcpy (xteds.xTEDS, "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS version= \"2.0
\n name= \"Drange_and_Curve_xTEDS \"> \n \t<Application name= \"Drange/Curve_FaultChecker \"
kind= \"data \"/> \n \t<Interface name= \"Drange_Interface \" id= \"1 \"> \n \t\t<Variable name= \"data \"
format= \"UINT08 \" kind= \"data \"> \n \t\t\t<Drange name= \"test1 \"> \n \t\t\t\t<Option name= \"hello \"
value= \"don't know \" alarm= \"help \" description= \"This is a test \"/> \n \t\t\t\t</Drange> \n \t\t\t\t</Variable>
\n \t\t\t\t<Notification> \n \t\t\t\t\t<DataMsg name= \"drange_all \" id= \"1 \" msgArrival= \"PERIODIC \"
msgRate= \"1 \"> \n \t\t\t\t\t\t<VariableRef name= \"data \"/> \n \t\t\t\t\t\t</DataMsg> \n \t\t\t\t\t\t</Notification> \n
\t\t\t\t\t\t</Interface> \n \t\t\t\t\t\t<Interface name= \"Curve_Interface \" id= \"2 \"> \n \t\t\t\t\t\t\t<Variable name= \"data \"
format= \"UINT16 \" kind= \"data \"> \n \t\t\t\t\t\t\t\t<Qualifier name= \"Qual1 \" value= \"1 \"/> \n \t\t\t\t\t\t\t\t\t<Curve
name= \"Test2 \"> \n \t\t\t\t\t\t\t\t\t\t<Coef value= \"nope \" description= \"This is another test \" exponent= \"yep
\"/> \n \t\t\t\t\t\t\t\t\t\t\t</Curve> \n \t\t\t\t\t\t\t\t\t\t\t\t</Variable> \n \t\t\t\t\t\t\t\t\t\t\t\t<Notification> \n \t\t\t\t\t\t\t\t\t\t\t\t\t<DataMsg name= \"curve_all \" id= \"1 \"
msgArrival= \"PERIODIC \" msgRate= \"1 \"> \n \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<VariableRef name= \"data \"/> \n \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t</DataMsg> \n \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t</Notification> \n \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t</Interface> \n</xTEDS> \n"); // set xTEDS

```

```
151:
152:  xteds.source.setSensorID(1);          // set the id of this application
153:  xteds.source.setPort(my_port);
154:  printf("Registering producer xTEDS on port %ld \n",my_port);
155:  xteds.Send();                        // register with the SDM
156: }
157:
158: void CancelxTEDS()
159: {
160:     SDMCancelxTEDS cancel;
161:     printf("Canceling xTEDS \n");
162:     cancel.source.setSensorID(1);
163:     cancel.source.setPort(my_port);
164:     cancel.Send();
165: }
166:
```


File: sdm/app/test/magtest.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMSubreqst.h"
3: #include "../common/message/SDMDeletesub.h"
4: #include "../common/message/SDMCancelxTEDS.h"
5: #include "../common/SubscriptionManager/SubscriptionManager.h"
6: #include "../common/MessageManager/MessageManager.h"
7: #include <string.h>
8: #include <sys/types.h>
9: #include <sys/stat.h>
10: #include <fcntl.h>
11: #include <unistd.h>
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <pthread.h>
15:
16: void RegisterxTEDS();
17: void CancelxTEDS();
18: void* Publisher(void *);
19: void* Listener(void *);
20:
21: SubscriptionManager subscriptions;
22: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER;
23: long my_port;
24:
25: int main(int argc, char** argv)
26: {
27:     pthread_t ListenerThread;
28:     pthread_t PublisherThread;
29:
30:     SDMInit(argc, argv);
31:     my_port = getPort();
32:     pthread_create(&ListenerThread, NULL, &Listener, NULL);
33:     RegisterxTEDS();
34:     pthread_create(&PublisherThread, NULL, &Publisher, NULL);
35:     pthread_join(PublisherThread, NULL);
36:     CancelxTEDS();
37:     pthread_cancel(ListenerThread);
38:     pthread_join(ListenerThread, NULL);
39: }
```

```

40:
41: void* Publisher(void * args)
42: {
43:     int published = 0;
44:     char data[20];
45:     double timestamp = 1;
46:     float x = 1.0;
47:     float y = 2.0;
48:     float z = 3.0;
49:     while(published < 100)
50:     {
51:         memcpy(data,&timestamp,8);
52:         x = 1.31*(float)(rand()&0xFFFF);
53:         y = 2.35*(float)(rand()&0xFFFF);
54:         z = 3.21*(float)(rand()&0xFFFF);
55:         printf("time:%f \tX:%f \tY:%f \tZ:%f \t \n",timestamp,x,y,z);
56:         memcpy(&data[8],&x,4);
57:         memcpy(&data[12],&y,4);
58:         memcpy(&data[16],&z,4);
59:         pthread_mutex_lock(&subscription_mutex);
60:         if (subscriptions.Publish(1,126,data,20)) published++;
61:         pthread_mutex_unlock(&subscription_mutex);
62:         sleep(1);
63:         timestamp++;
64:     }
65:     return NULL;
66: }
67:
68: void* Listener(void * args)
69: {
70:     char buf[BUFSIZE];
71:     SDMSubreqst sub;
72:     SDMDeletesub del;
73:     MessageManager mm;
74:     mm.Async_Init(my_port);
75:     while(1)
76:     {
77:         pthread_testcancel();
78:         if(mm.IsReady())
79:         {
80:             switch(mm.GetMessage(buf))

```

```

81:     {
82:     case SDM_Subreqst:
83:         sub.Unmarshal(buf);
84:         pthread_mutex_lock(&subscription_mutex);
85:         subscriptions.AddSubscription(sub);
86:         pthread_mutex_unlock(&subscription_mutex);
87:         break;
88:     case SDM_Deletesub:
89:         del.Unmarshal(buf);
90:         pthread_mutex_lock(&subscription_mutex);
91:         subscriptions.RemoveSubscription(del);
92:         pthread_mutex_unlock(&subscription_mutex);
93:         break;
94:     default:
95:         printf("Invalid Message found! \n");
96:         fflush(NULL);
97:         break;
98:     }
99: }
100: else
101: {
102:     usleep(100000);
103: }
104: }
105: return NULL;
106: }
107:
108: void RegisterxTEDS()
109: {
110:     SDMxTEDS xteds; // create an xTEDS registration message
111:
112:     strcpy(xteds.xTEDS, "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<!--Sample XML file
generated by XMLSpy v2005 rel. 3 U (http://www.altova.com)--> \n<xTEDS xmlns=
\"http://www.interfacecontrol.com/SPA/xTEDS \" xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-
instance \" xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS xTEDS.xsd \" name=
\"String \" version= \"2.0 \" description= \"Text \"> \n \t<Device name= \"BillingsleyTFM100S \" id=
\"123 \" kind= \"Magnetometer \" qualifier= \"3 AXIS \" description= \"Billingsley TFM100S
Magnetometer \" powerRequirements= \".6 \" directionXYZ= \"Text \" referenceFrequency= \"Text \"
sensitivityAtReference= \"Text \" calDueDate= \"2006-05-25 \" referenceTemperature= \"Text \"
serialNumber= \"Text \" manufacturerId= \"Billingsley \" electricalOutput= \"Text \"
measurementRange= \"Text \" qualityFactor= \"Text \" versionLetter= \"S \" modelId= \"TFM100 \"
calibrationDate= \"2005-05-25 \" temperatureCoefficient= \"Text \" componentKey= \"MagTest \"
SPA_U_hub= \"PanelNegZ_HubA \" SPA_U_port= \"1 \"/> \n \t<Interface name=

```

```

\MAGNETOMETER_INTERFACE \" id= \"1 \"> \n \t<!-- raw counts --> \n \t \t<Variable name=
\"Field_X_Raw \" kind= \"Magnetic_Field \" qualifier= \"X Axis Device Frame \" description= \"Raw
Voltage from Magnetic Field X Sensor Frame \" accuracy= \"0.005 \" defaultValue= \"0 \" format=
\"UINT16 \" length= \"1 \" units= \"Counts \" precision= \"2 \" rangeMin= \"0 \" rangeMax= \"10 \"/> \n \t
\t<Variable name= \"Field_Y_Raw \" kind= \"Magnetic_Field \" qualifier= \"Y Axis Device Frame \"
description= \"Raw Voltage from Magnetic Field Y Sensor Frame \" accuracy= \"0.005 \" defaultValue=
\"0 \" format= \"UINT16 \" length= \"1 \" units= \"Counts \" precision= \"2 \" rangeMin= \"0 \"
rangeMax= \"10 \"/> \n \t \t<Variable name= \"Field_Z_Raw \" kind= \"Magnetic_Field \" qualifier= \"Z
Axis Device Frame \" description= \"Raw Voltage from Magnetic Field Z Sensor Frame \" accuracy=
\"0.005 \" defaultValue= \"0 \" format= \"UINT16 \" length= \"1 \" units= \"Counts \" precision= \"2 \"
rangeMin= \"0 \" rangeMax= \"10 \"/> \n \t<!-- Gauss --> \n \t \t<Variable name= \"Field_X \" kind=
\"Magnetic_Field \" qualifier= \"X Axis Device Frame \" description= \"Magnetic Field X Sensor Frame
\" accuracy= \"0.005 \" defaultValue= \"0.0 \" format= \"FLOAT32 \" length= \"1 \" units= \"Gauss \"
precision= \"2 \" rangeMin= \"0 \" rangeMax= \"10.0 \"/> \n \t \t<Variable name= \"Field_Y \" kind=
\"Magnetic_Field \" qualifier= \"Y Axis Device Frame \" description= \"Magnetic Field Y Sensor Frame
\" accuracy= \"0.005 \" defaultValue= \"0.0 \" format= \"FLOAT32 \" length= \"1 \" units= \"Gauss \"
precision= \"2 \" rangeMin= \"0 \" rangeMax= \"10.0 \"/> \n \t \t<Variable name= \"Field_Z \" kind=
\"Magnetic_Field \" qualifier= \"Z Axis Device Frame \" description= \"Magnetic Field Z Sensor Frame \"
accuracy= \"0.005 \" defaultValue= \"0.0 \" format= \"FLOAT32 \" length= \"1 \" units= \"Gauss \"
precision= \"2 \" rangeMin= \"0 \" rangeMax= \"10.0 \"/> \n \t<!-- data rate --> \n \t \t<Variable name=
\"Data_Rate \" kind= \"Magnetic_Field \" qualifier= \"Message Rate \" description= \"Data rate \"
accuracy= \"1 \" defaultValue= \"1 \" format= \"UINT16 \" length= \"1 \" units= \"Hz \" precision= \"1 \"
rangeMin= \"1 \" rangeMax= \"1000 \"/> \n \t<!-- calibration --> \n \t \t<Variable name=
\"Calibration_CurveX \" kind= \"Magnetic_Field \" qualifier= \"Polynomial \" description= \"Cal Curve X
Polynomial \" defaultValue= \"0,1,0 \" format= \"FLOAT32 \" length= \"3 \" precision= \"3 \"/> \n \t
\t<Variable name= \"Calibration_CurveY \" kind= \"Magnetic_Field \" qualifier= \"Polynomial \"
description= \"Cal Curve Y Polynomial \" defaultValue= \"0,1,0 \" format= \"FLOAT32 \" length= \"3 \"
precision= \"3 \"/> \n \t \t<Variable name= \"Calibration_CurveZ \" kind= \"Magnetic_Field \" qualifier=
\"Polynomial \" description= \"Cal Curve Z Polynomial \" defaultValue= \"0,1,0 \" format= \"FLOAT32
\" length= \"3 \" precision= \"3 \"/> \n \t<!-- timestamp --> \n \t \t<Variable name= \"Timestamp \" kind=
\"Magnetic_Field \" qualifier= \"Time \" description= \"Sample Time \" accuracy= \"0.001 \"
defaultValue= \"0.0 \" format= \"FLOAT64 \" length= \"1 \" units= \"seconds \" precision= \"2 \"
rangeMin= \"0 \"/> \n \t<!-- request messages --> \n \t \t<Request> \n \t \t \t<CommandMsg name=
\"SetCalibration \" description= \"Text \" id= \"123 \"> \n \t \t \t \t<VariableRef name=
\"Calibration_CurveX \"/> \n \t \t \t \t<VariableRef name= \"Calibration_CurveY \"/> \n \t \t \t \t
\t<VariableRef name= \"Calibration_CurveZ \"/> \n \t \t \t</CommandMsg> \n \t \t \t<DataReplyMsg
name= \"CalibrationData \" description= \"TFM100S Calibration Data \" id= \"127 \"> \n \t \t \t \t
\t<VariableRef name= \"Timestamp \"/> \n \t \t \t \t<VariableRef name= \"Calibration_CurveX \"/> \n \t \t
\t \t<VariableRef name= \"Calibration_CurveY \"/> \n \t \t \t \t<VariableRef name= \"Calibration_CurveZ
\"/> \n \t \t \t</DataReplyMsg> \n \t \t</Request> \n \t<!-- notification messages --> \n \t \t<Notification>
\n \t \t \t<DataMsg name= \"RawData \" description= \"TFM100S Magnetometer Raw Counts \"
msgArrival= \"PERIODIC \" id= \"125 \"> \n \t \t \t \t<VariableRef name= \"Timestamp \"/> \n \t \t \t \t
\t<VariableRef name= \"Field_X_Raw \"/> \n \t \t \t \t<VariableRef name= \"Field_Y_Raw \"/> \n \t \t \t
\t<VariableRef name= \"Field_Z_Raw \"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t
\t<Notification> \n \t \t<DataMsg name= \"MagFieldData \" description= \"TFM100S Magnetometer
Fields \" msgArrival= \"PERIODIC \" id= \"126 \"> \n \t \t \t \t<VariableRef name= \"Timestamp \"/> \n \t
\t \t \t<VariableRef name= \"Field_X \"/> \n \t \t \t \t<VariableRef name= \"Field_Y \"/> \n \t \t \t
\t<VariableRef name= \"Field_Z \"/> \n \t \t \t</DataMsg> \n \t \t</Notification> \n \t<!-- command
messages --> \n \t \t<Command> \n \t \t \t<CommandMsg name= \"SetDataRate \" description= \"Text \"
id= \"124 \"> \n \t \t \t \t<VariableRef name= \"Data_Rate \"/> \n \t \t \t</CommandMsg> \n \t
\t</Command> \n \t</Interface> \n</xTEDS>"); // set xTEDS

```

```

113:
114:  xteds.source.setSensorID(1);          // set the id of this application
115:  xteds.source.setPort(my_port);
116:  printf("Registering producer xTEDS on port %ld \n",my_port);
117:  xteds.Send();                        // register with the SDM
118: }
119:
120: void CancelxTEDS()
121: {
122:     SDMCancelxTEDS cancel;
123:     printf("Canceling xTEDS \n");
124:     cancel.source.setSensorID(1);
125:     cancel.source.setPort(my_port);
126:     cancel.Send();
127: }

```

File: sdm/app/test/messagecountconsumer.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMCancel.h"
7: #include "../common/MessageManipulator/MessageManipulator.h"
8: #include "../common/MessageManager/MessageManager.h"
9: #include "../common/marshall.h"
10:
11: #include <string.h>
12: #include <unistd.h>
13: #include <stdio.h>
14: #include <sys/types.h>
15: #include <sys/wait.h>
16: #include <signal.h>
17:
18: #define DATA_PROVIDER    1
19:
20: long my_port;
21: SDMConsume consume_msgs[4];
22: SDMComponent_ID service_provider;
23: static int providerIndex = -1;
24:
25: MessageManipulator data_manipulator;
26:
27: void DataHandler(SDMDData& dat,long length)
28: {
29: unsigned int total_recd = 0;
30: unsigned int lastsec_recd = 0;
31: unsigned int total_sent = 0;
32: unsigned int lastsec_sent = 0;
33:
34: total_recd = GET_UINT(&dat.msg[0]);
35: lastsec_recd = GET_UINT(&dat.msg[4]);
36: total_sent = GET_UINT(&dat.msg[8]);
37: lastsec_sent = GET_UINT(&dat.msg[12]);
38:
39:
```

```

40: if(dat.source.getPort() == PORT_DM)
41: {
42:     printf("DM :: Total messages recd = %u, Messages last second recd= %u, Total messages sent =
%u, Messages last second sent = %u \n", total_recd, lastsec_recd, total_sent, lastsec_sent);
43: }
44: else if (dat.source.getPort() == PORT_TM)
45:     {
46:     printf("TM :: Total messages recd = %u, Messages last second recd= %u, Total messages sent =
%u, Messages last second sent = %u \n", total_recd, lastsec_recd, total_sent, lastsec_sent);
47:     }
48: else if (dat.source.getPort() == PORT_PM)
49: {
50:     printf("PM :: Total messages recd = %u, Messages last second recd= %u, Total messages sent =
%u, Messages last second sent = %u \n", total_recd, lastsec_recd, total_sent, lastsec_sent);
51: }
52: else if (dat.source.getPort() == PORT_SM)
53: {
54:     printf("SM :: Total messages recd = %u, Messages last second recd= %u, Total messages sent =
%u, Messages last second sent = %u \n", total_recd, lastsec_recd, total_sent, lastsec_sent);
55: }
56: else
57:     printf("Unknown sender. %ld:%hd \n", dat.source.getAddress(), dat.source.getPort());
58: }
59:
60: void RegInfoHandler(SDMRegInfo& info)
61: {
62: providerIndex++;
63: //Set the port we will be receiving on
64: consume_msgs[providerIndex].destination.setPort(my_port);
65: //copy the sensor id into the consume message
66: consume_msgs[providerIndex].source = info.source;
67: //copy the msg id into the consume message
68: consume_msgs[providerIndex].msg_id = info.msg_id;
69:
70: if (info.source == DataManager)
71:     printf("RegInfo for DataManager received.");
72: else if (info.source == TaskManager)
73:     printf("RegInfo for TaskManager received.");
74: //else if (info.source == SensorManager)
75: // printf("RegInfo for SensorManager received.");
76:

```

```

77: printf("Message      id      is      %ud      interface      id      is      %ud
\n",info.msg_id.getMessage(),info.msg_id.getInterface());
78: fflush(NULL);
79: switch(info.id)
80: {
81: case DATA_PROVIDER:
82:     if (info.type != 1)
83:     {
84:         data_manipulator.setMsgDef(info.msg_def);
85:         printf("New Data provider found");
86:         printf(" \n \n%s \n",info.msg_def);
87:         //Send the consume message
88:         consume_msgs[providerIndex].Send();
89:     }
90:     break;
91:
92: }
93:
94: }
95:
96: void SigHandler(int signum)
97: {
98: if (signum == SIGINT)
99: {
100:     SDMCancel cancel_msg;
101:     cancel_msg.destination.setPort(my_port);
102:     printf("Cancelling subscriptions...");
103:     for (int i = 0; i <= providerIndex; i++)
104:     {
105:         cancel_msg.source = consume_msgs[i].source;
106:         cancel_msg.msg_id = consume_msgs[i].msg_id;
107:         cancel_msg.Send();
108:     }
109:     printf("Done. \n");
110:     _exit(0);
111: }
112: }
113: int main(int argc,char** argv)
114: {
115:     MessageManager mm;
116:     SDMData dat;

```



```

117:  SDMRegInfo info;
118:  SDMReqReg req_reg;
119:  char buf[BUFSIZE];
120:  long length;
121:
122:  //initialize consumer
123:  SDMInit(argc,argv);
124:  my_port = getPort();
125:  mm.Async_Init(my_port);
126:  signal(SIGINT, SigHandler);
127:
128:  printf("%s listening on port %ld \n",argv[0],my_port);
129:
130:  while(1)
131:  {
132:      if (mm.IsReady())
133:      {
134:          switch(mm.GetMessage(buf,length))
135:          {
136:              case SDM_Data:
137:                  dat.Unmarshal(buf,length);
138:                  DataHandler(dat,length);
139:                  break;
140:              case SDM_RegInfo:
141:                  if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)
142:                  {
143:                      RegInfoHandler(info);
144:                  }
145:                  break;
146:              default:
147:                  printf("Unexpected message \n");
148:          }
149:      }
150:      else
151:      { //check for data and service providers
152:          if(consume_msgs[0].source.getSensorID() == 0)
153:          {
154:              //request info on integer data providers
155:              //Set variable name
156:              strcpy(req_reg.item_name,"Total_Messages_Recd");
157:              //Set the quallist can be empty

```

```

158:         strcpy(req_reg.quallist,"< format= \"UINT32 \"/>");
159:         req_reg.reply
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
160:         req_reg.destination.setPort(my_port);
161:         req_reg.id = DATA_PROVIDER;
162:         req_reg.Send();
163:         printf("Searching for new data provider \n");
164:         sleep(2);
165:     }
166:     usleep(1000);
167: }
168: }
169: }

```

File: sdm/app/test/VarInfoParserTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <stdlib.h>
4: #include "../common/VarInfoParser/VarInfoParser.h"
5:
6: int main(int argc, char ** argv)
7: {
8:     VarInfoParser parser;
9:     char variable[] = "<Variable format= \"UINT08 \" kind= \"Component \" name= \"ComponentKey \"
length= \"129 \" description= \"The component key for a device \" rangeMin= \"3.2 \" rangeMax= \"5.5 \"
defaultValue= \"df1 \" precision= \".0001 \" units= \"text \" accuracy= \".01011 \" scaleFactor= \"3.2 \"
scaleUnits= \"sU \"> \n \t<Qualifier name= \"Qual1 \" value= \"13 \"/> \n \t<Qualifier name= \"Qual2 \"
value= \"14 \"/> \n \t<Qualifier name= \"Qual3 \" value= \"15 \"/> \n \t<Qualifier name= \"Qual4 \"
value= \"16 \"/> \n \t<Drange name= \"amActivityPriorityEnum \"> \n \t \t<Option value= \"0 \" name=
\"LOW \"/> \n \t \t<Option value= \"1 \" name= \"MEDIUM \"/> \n \t \t<Option value= \"2 \" name=
\"HIGH \"/> \n \t</Drange> \n</Variable>";
10: //char variable[] = "<Variable name= \"amActivityPriority \" kind= \"tbd \" format= \"UINT16
\"></Variable>";
11: //char variable[] = " <Variable name= \"myVar \" kind= \"tbd \" format= \"UINT16 \"> <Qualifier
name= \"Qual1 \" value= \"13 \"/> <Qualifier name= \"Qual2 \" value= \"14 \"/> <Qualifier name=
\"Qual3 \" value= \"15 \"/> <Qualifier name= \"Qual4 \" value= \"16 \"/> </Variable> ";
12:
13: printf("Parsing this variable definition: \n%s \n \n",variable);
14:
15: if (parser.setVarInfo(variable))
16:     printf("Parse succeeded. \n");
17: else
18:     printf("Parse failed. \n");
19: //
20: //Output the info about the variable
21: //
22: char buf[128];
23: if (parser.getName(buf))
24:     printf("Name is %s \n",buf);
25: if (parser.getLength(buf))
26:     printf("Length is %s \n",buf);
27: if (parser.getKind(buf))
28:     printf("Kind is %s \n",buf);
29: if (parser.getId(buf))
30:     printf("Id is %s \n",buf);
31: if (parser.getRangeMax(buf))
```

```

32:     printf("RangeMax is %s \n",buf);
33: if (parser.getRangeMin(buf))
34:     printf("RangeMin is %s \n",buf);
35: if (parser.getDefaultValue(buf))
36:     printf("Default value is %s \n",buf);
37: if (parser.getPrecision(buf))
38:     printf("Precision is %s \n",buf);
39: if (parser.getUnits(buf))
40:     printf("Units is %s \n",buf);
41: if (parser.getAccuracy(buf))
42:     printf("Accuracy is %s \n",buf);
43: if (parser.getScaleFactor(buf))
44:     printf("Scale factor is %s \n",buf);
45: if (parser.getScaleUnits(buf))
46:     printf("Scale units is %s \n",buf);
47: if (parser.getFormat(buf))
48:     printf("Format is %s \n",buf);
49: if (parser.getDescription(buf))
50:     printf("Description is %s \n",buf);
51: //
52: //Check for qualifiers
53: //
54: if (parser.hasQualifiers())
55: {
56:     char value[33];
57:     parser.getQualValueByName(value,"Qual1");
58:     printf("Value associated with Qual1 is %s \n",value);
59:     parser.getQualValueByName(value,"Qual2");
60:     printf("Value associated with Qual2 is %s \n",value);
61:     parser.getQualValueByName(value,"Qual3");
62:     printf("Value associated with Qual3 is %s \n",value);
63:     parser.getQualValueByName(value,"Qual4");
64:     printf("Value associated with Qual4 is %s \n",value);
65: }
66: if (parser.getDrangeSize() > 0)
67: {
68:     printf("Variable has %d option elements \n",parser.getDrangeSize());
69:     char nameBuf[33];
70:     char valueBuf[33];
71:     //Loop going under/over the bounds of the range to assure the function won't return anything
72:     for (int i = -3; i <= parser.getDrangeSize()+3; i++)

```

```
73:     {
74:         if (parser.getOptionNumber(i, nameBuf, valueBuf))
75:             printf("Option %d - Name is %s value is %s \n", i, nameBuf, valueBuf);
76:     }
77: }
78: //Re-parse if running valgrind to check for memory leaks
79: parser.setVarInfo(variable);
80:
81: return 0;
82: }
```

File: sdm/app/test/ServicePID.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMSubreqst.h"
3: #include "../common/message/SDMDeletesub.h"
4: #include "../common/message/SDMCancelxTEDS.h"
5: #include "../common/message/SDMService.h"
6: #include "../common/message/SDMData.h"
7: #include "../common/message/SDMConsume.h"
8: #include "../common/MessageManager/MessageManager.h"
9: #include <string.h>
10: #include <sys/types.h>
11: #include <sys/stat.h>
12: #include <fcntl.h>
13: #include <unistd.h>
14: #include <stdio.h>
15: #include <stdlib.h>
16: #include <pthread.h>
17:
18: void RegisterxTEDS();
19: void CancelxTEDS();
20:
21: long my_port;
22:
23: int main(int argc, char** argv)
24: {
25:     SDMService service;
26:     SDMConsume con;
27:     char buf[BUFSIZE];
28:     long length;
29:     MessageManager mm;
30:     bool one = false;
31:
32:     int pid = 0;
33:     int sid = 0;
34:     SDMData dat;
35:
36:     SDMInit(argc, argv);
37:     my_port = getPort();
38:     mm.Async_Init(my_port);
39:     RegisterxTEDS();
```

```

40: sleep(1);
41: service.source.setSensorID(1);
42: service.command_id = 264;
43: service.destination.setPort(my_port);
44: service.length = 4;
45: pid = atoi(argv[3]);
46: memcpy(service.data,&pid,4);
47: service.Send();
48: while(1)
49: {
50:     if (mm.IsReady())
51:     {
52:         mm.GetMessage(buf,length);
53:         dat.Unmarshal(buf,4);
54:         switch(dat.msg_id.getInterfaceMessagePair())
55:         {
56:             case 257:
57:                 memset(&pid,0,4);
58:                 memcpy(&pid,dat.msg,4);
59:                 if(one == false)
60:                 {
61:                     printf("Registered SensorID's are %d",pid);
62:                     fflush(NULL);
63:                     one = true;
64:                 }
65:             else
66:             {
67:                 printf(", %d",pid);
68:                 fflush(NULL);
69:                 if(pid == sid)
70:                 {
71:                     printf(" \n");
72:                     CancelxTEDS();
73:                     return 0;
74:                 }
75:             }
76:             break;
77:             case 265:
78:                 memcpy(&sid,&dat.msg[0],4);
79:                 printf("My SensorID is %d \n",sid);
80:                 con.source.setSensorID(1);

```

```

81:         con.msg_id = 257;
82:         con.destination.setPort(my_port);
83:         con.Send();
84:         break;
85:     default:
86:         printf("Unexpected data message \n");
87:         break;
88:     }
89: }
90: }
91: }
92:
93: void RegisterxTEDS()
94: {
95:     SDMxTEDS xteds; // create an xTEDS registration message
96:
97:     strcpy (xteds.xTEDS,"<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS version= \"2.0 \"
name= \"Producer_xTEDS \"> \n \t<Application name= \"producer \" kind= \"data \"/> \n \t<Interface
name= \"Producer_Interface \" id= \"1 \"> \n \t<Variable name= \"data \" format= \"UINT16 \" kind=
\"data \" qualifier= \"Single Value \"/> \n \t<Notification> \n \t \t<DataMsg name= \"all \" id= \"1 \"
msgArrival= \"PERIODIC \" msgRate= \"1 \"> \n \t \t \t<VariableRef name= \"data \"/> \n \t
\t</DataMsg> \n \t</Notification> \n \t</Interface> \n</xTEDS> \n"); // set xTEDS
98:
99:     xteds.source.setSensorID(1); // set the id of this application
100:     xteds.source.setPort(my_port);
101:     printf("Registering producer xTEDS on port %ld \n",my_port);
102:     xteds.Send(); // register with the SDM
103: }
104:
105: void CancelxTEDS()
106: {
107:     SDMCancelxTEDS cancel;
108:     printf("Canceling xTEDS \n");
109:     cancel.source.setSensorID(1);
110:     cancel.source.setPort(my_port);
111:     cancel.Send();
112: }

```


File: sdm/app/test/ClassTests/ProviderSubscriptionListTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5:
6: #define DEBUG_SUB_LIST 1
7: #include "../dm/ProviderSubscriptionList.h"
8: #include "../common/message/SDMComponent_ID.h"
9: #include "../common/message/SDMMessage_ID.h"
10:
11: int main(int argc, char** argv)
12: {
13:     const int SIZE_IDS = 10;
14:     SDMComponent_ID pids[SIZE_IDS];
15:     SDMComponent_ID sids[SIZE_IDS];
16:     SDMMMessage_ID mids[SIZE_IDS];
17:
18:     for (int i = 0; i < SIZE_IDS; i++)
19:     {
20:         pids[i].setSensorID(i + 10);
21:         sids[i].setSensorID(i);
22:         mids[i].setMessage(i);
23:     }
24:
25:     ProviderSubscriptionList list;
26:
27:     for (int i = 0; i < SIZE_IDS; i++)
28:         list.Add(pids[i], sids[i], mids[i]);
29:
30:     printf("Adding one subscriber to all providers \n \n");
31:     // Add one subscriber to all providers
32:     for (int i = 0; i < SIZE_IDS; i++)
33:         list.Add(pids[i], sids[0], mids[i]);
34:
35:     printf("Adding one provider to all subscribers \n \n");
36:     // Add one provider to all subscribers
37:     for (int i = 0; i < SIZE_IDS; i++)
38:         list.Add(pids[0], sids[i], mids[i]);
39:
```

```
40: printf("Removing the one subscriber \n \n");
41: list.SubscriberFinish(sids[0]);
42:
43: printf("Removing the one provider \n \n");
44: list.ProviderFinish(pids[0]);
45:
46:
47: return 0;
48: }
```

File: sdm/app/test/ClassTests/SubManTest.cpp

```
1: #include <sys/socket.h>
2: #include <netinet/in.h>
3: #include <arpa/inet.h>
4: #include <stdio.h>
5: #include "../common/SubscriptionManager/SubscriptionManager.h"
6: #include "../common/message/SDMSubreqst.h"
7: #include "../common/message/SDMDeletesub.h"
8: #include "../common/message/SDMSerreqst.h"
9:
10:
11: int main(int argc, char ** argv)
12: {
13:     SubscriptionManager sm;
14:     SDMSubreqst sub_msg;
15:     SDMSerreqst ser_msg;
16:     SDMDeletesub del_msg;
17:
18:     sub_msg.msg_id.setInterface(1);
19:     sub_msg.msg_id.setMessage(2);
20:     sub_msg.destination.setAddress(inet_addr("127.0.0.1"));
21:     sub_msg.destination.setPort(4050);
22:     sm.AddSubscription(sub_msg) ? printf("First added \n") : printf("First not added \n");
23:     sm.AddSubscription(sub_msg) ? printf("Second added \n") : printf("Second not added \n");
24:
25:     ser_msg.reply_id.setInterface(1);
26:     ser_msg.reply_id.setMessage(3);
27:     sub_msg.destination.setAddress(inet_addr("127.0.0.1"));
28:     sub_msg.destination.setPort(4050);
29:     sm.AddSubscription(ser_msg) ? printf("Third added \n") : printf("Third not added \n");
30:     sub_msg.destination.setPort(4051);
31:     sm.AddSubscription(ser_msg) ? printf("Fourth added \n") : printf("Fourth not added \n");
32:
33:     sm.Publish(1,3,"A",1);
34:     return 0;
35: }
```

File: sdm/app/test/ClassTests/MessageManipulatorTest.cpp

```
1: #include "../common/MessageManipulator/MessageManipulator.h"
2: #include "../common/MessageManager/MessageManager.h"
3: #include "../common/message/SDMxTEDS.h"
4: #include "../common/message/SDMReqReg.h"
5: #include "../common/message/SDMRegInfo.h"
6: #include "../DMTests/xTEDSPoster.h"
7:
8: #include <string.h>
9: #include <unistd.h>
10: #include <stdio.h>
11: #include <sys/types.h>
12: #include <sys/wait.h>
13: #include <signal.h>
14:
15: // Message enumeration values
16: enum { NDMT=0, NDMTI=1, CCMT=2, CCMTI=3, CFMT=4, CFMTI=5, RCMT=6, RCMTI=7,
RDRMT=8, RDRMTI=9, RFMT=10, RFMTI=11 };
17:
18: // Values to get/set into message buffers
19: const unsigned char UCHAR_VAL = 0x51;
20: const char CHAR_VAL = 0x77;
21: const unsigned short USHORT_VAL = 0x3432;
22: const short SHORT_VAL = 0x7865;
23: const unsigned long UINT_VAL = 0x89473244;
24: const long INT_VAL = 0x75884763;
25: const float FLOAT_VAL = 3.14159;
26: const double DOUBLE_VAL = 4.545586599;
27:
28: const unsigned char UCHAR_INVALID = 13;
29: const char CHAR_INVALID = 13;
30: const unsigned short USHORT_INVALID = 13;
31: const short SHORT_INVALID = 13;
32: const unsigned long UINT_INVALID = 13;
33: const long INT_INVALID = 13;
34: const float FLOAT_INVALID = 13.0f;
35: const double DOUBLE_INVALID = 13.0;
36:
37: const long NUM_TESTS = 12;
38: MessageManipulator Manipulators[NUM_TESTS];
```

```

39:
40: const long myPort = 4256;
41: bool VerifyBufferData(char* Buffer);
42: bool VerifyBufferDataFault(char* Buffer);
43: bool CmdMsgSetAll (SDMCommand &cmd, MessageManipulator& mm);
44: bool CmdMsgSetAllInvalid (SDMCommand &cmd, MessageManipulator& mm);
45: bool CmdMsgGetAll (SDMCommand& cmd, MessageManipulator& mm);
46: bool CmdMsgGetAllInvalid (SDMCommand& cmd, MessageManipulator& mm);
47: bool CmdMsgSetArray(SDMCommand& cmd, MessageManipulator& mm);
48:
49: bool DataMsgSetAll (SDMData &dat, MessageManipulator& mm, enum msg_type type);
50: bool DataMsgSetAllInvalid (SDMData &dat, MessageManipulator& mm, enum msg_type type);
51: bool DataMsgGetAll (SDMData& dat, MessageManipulator& mm, enum msg_type type);
52: bool DataMsgGetAllInvalid (SDMData& dat, MessageManipulator& mm, enum msg_type type);
53:
54: bool SerMsgSetAll (SDMService &ser, MessageManipulator& mm);
55: bool SerMsgSetAllInvalid (SDMService &ser, MessageManipulator& mm);
56: bool SerMsgGetAll (SDMSerreqst& ser, MessageManipulator& mm);
57: bool SerMsgGetAllInvalid (SDMSerreqst& ser, MessageManipulator& mm);
58:
59: int main(int argc,char** argv)
60: {
61:     SDMInit(argc, argv);
62:     xTEDSPoster poster;
63:     if (poster.PostxTEDS("MessageManipulatorTest.xml") < 0)
64:     {
65:         printf("Could not post xTEDs. \n");
66:         return 0;
67:     }
68:
69:     MessageManager mm;
70:     mm.Async_Init(myPort);
71:
72:     printf("Requesting messages...");
73:     fflush(NULL);
74:     SDMReqReg req;
75:     req.destination.setPort(myPort);
76:     strcpy(req.device,"MessageManipulatorTest");
77:     strcpy(req.item_name,"NotDatMsgTest");
78:     req.Send();
79:

```

```
80: strcpy(req.item_name,"CmdCmdMsgTest");
81: req.id = CCMT;
82: req.Send();
83:
84: strcpy(req.item_name,"CmdFaultMsgTest");
85: req.id = CFMT;
86: req.Send();
87:
88: strcpy(req.item_name,"ReqCmdMsgTest");
89: req.id = RCMT;
90: req.Send();
91:
92: strcpy(req.item_name,"ReqDataRplyMsgTest");
93: req.id = RDRMT;
94: req.Send();
95:
96: strcpy(req.item_name,"ReqFaultMsgTest");
97: req.id = RFMT;
98: req.Send();
99:
100:  strcpy(req.item_name,"NotDatMsgTestInvalid");
101:  req.id = NDMTI;
102:  req.Send();
103:
104:  strcpy(req.item_name,"CmdCmdMsgTestInvalid");
105:  req.id = CCMTI;
106:  req.Send();
107:
108:  strcpy(req.item_name,"CmdFaultMsgTestInvalid");
109:  req.id = CFMTI;
110:  req.Send();
111:
112:  strcpy(req.item_name,"ReqCmdMsgTestInvalid");
113:  req.id = RCMTI;
114:  req.Send();
115:
116:  strcpy(req.item_name,"ReqDataRplyMsgTestInvalid");
117:  req.id = RDRMTI;
118:  req.Send();
119:
120:  strcpy(req.item_name,"ReqFaultMsgTestInvalid");
```

```

121: req.id = RFMTI;
122: req.Send();
123:
124: int count = 0;
125: char buf[BUFSIZE];
126: SDMRegInfo info;
127: while (count < NUM_TESTS)
128: {
129:     switch(mm.BlockGetMessage(buf))
130:     {
131:     case SDM_RegInfo:
132:         if (info.Unmarshal(buf) > 0)
133:         {
134:             printf("%d ",++count);
135:             fflush(NULL);
136:             if (Manipulators[info.id].setMsgDef(info.msg_def) == EMPTY)
137:             {
138:                 printf("%s \n",info.msg_def);
139:                 return 0;
140:             }
141:         }
142:         break;
143:     default:
144:         printf("Unexpected message received. \n");
145:         break;
146:     }
147: }
148: printf("...done. \n");
149:
150: //////////////////////////////////////
151: // Testing <Command>
152: //
153: printf("-----Testing with <Command><CommandMsg>----- \n");
154: SDMCommand cmd;
155: CmdMsgSetAll(cmd, Manipulators[CCMT]);
156: if (!VerifyBufferData(cmd.data))
157: {
158:     printf("Buffer mismatch. \n");
159:     return 0;
160: }
161: if ( !CmdMsgGetAll(cmd, Manipulators[CCMT]) ||

```

```

162:     !CmdMsgSetAllInvalid(cmd, Manipulators[CCMTI]) ||
163:     !CmdMsgGetAllInvalid(cmd, Manipulators[CCMTI]) )
164:     return 0;
165: if (!CmdMsgSetArray(cmd, Manipulators[CCMTI]) )
166:     return 0;
167: printf("-----<Command><CommandMsg> test done----- \n");
168: // Test functions for SDMDData
169: SDMDData dat;
170:
171: printf("-----Testing with <Command><FaultMsg>----- \n");
172: memset(dat.msg, 0, sizeof(dat.msg));
173:
174: printf("Testing FaultMsg using DATAMSG (should fail)... \n");
175: if (!DataMsgSetAll(dat, Manipulators[CCMT], DATAMSG))
176:     printf(".....passed. \n");
177: else
178: {
179:     printf("failed. \n"); return 0;
180: }
181:
182: if (!DataMsgSetAll(dat, Manipulators[CCMT], FAULTMSG))
183:     return 0;
184: if (!VerifyBufferDataFault(dat.msg))
185: {
186:     printf("Buffer mismatch. \n");
187:     return 0;
188: }
189: if ( !DataMsgGetAll(dat, Manipulators[CCMT], FAULTMSG) ||
190:     !DataMsgSetAllInvalid(dat, Manipulators[CCMTI], FAULTMSG) ||
191:     !DataMsgGetAllInvalid(dat, Manipulators[CCMTI], FAULTMSG) )
192:     return 0;
193: printf("-----<Command><FaultMsg> test done----- \n");
194: //
195: //</Command>
196: //////////////////////////////////////
197:
198: //////////////////////////////////////
199: // Testing <Notification>
200: //
201: printf("-----Testing with <DataMsg>----- \n");
202: memset(dat.msg, 0, sizeof(dat.msg));

```



```

203: DataMsgSetAll(dat, Manipulators[NDMT], DATAMSG);
204: if (!VerifyBufferData(dat.msg))
205: {
206:     printf("Buffer mismatch. \n");
207:     return 0;
208: }
209: if ( !DataMsgGetAll(dat, Manipulators[NDMT], DATAMSG) ||
210:     !DataMsgSetAllInvalid(dat, Manipulators[NDMTI], DATAMSG) ||
211:     !DataMsgGetAllInvalid(dat, Manipulators[NDMTI], DATAMSG) )
212:     return 0;
213: printf("-----<DataMsg> test done----- \n");
214:
215: printf("-----Testing with <FaultMsg>----- \n");
216: memset(dat.msg, 0, sizeof(dat.msg));
217: DataMsgSetAll(dat, Manipulators[NDMT], FAULTMSG);
218: if (!VerifyBufferDataFault(dat.msg))
219: {
220:     printf("Buffer mismatch. \n");
221:     return 0;
222: }
223: if ( !DataMsgGetAll(dat, Manipulators[NDMT], FAULTMSG) ||
224:     !DataMsgSetAllInvalid(dat, Manipulators[NDMTI], FAULTMSG) ||
225:     !DataMsgGetAllInvalid(dat, Manipulators[NDMTI], FAULTMSG) )
226:     return 0;
227: printf("-----<FaultMsg> test done----- \n");
228: //
229: //</Notification>
230: //////////////////////////////////////
231:
232: //////////////////////////////////////
233: // Testing <Request>
234: //
235: SDMServ serv;
236: SDMSerreqst serq;
237: printf("-----Testing with <Request><Command>----- \n");
238: memset(cmd.data, 0, sizeof(cmd.data));
239: if (!CmdMsgSetAll(cmd, Manipulators[RCMT]))
240:     return 0;
241: if (!VerifyBufferData(cmd.data))
242: {
243:     printf("Buffer mismatch. \n");

```

```

244:     return 0;
245: }
246: if ( !CmdMsgGetAll(cmd, Manipulators[RCMT]) ||
247:     !CmdMsgSetAllInvalid(cmd, Manipulators[RCMTI]) ||
248:     !CmdMsgGetAllInvalid(cmd, Manipulators[RCMTI]) )
249:     return 0;
250: printf("-----<Request><CommandMsg> test done----- \n");
251:
252: printf("-----Testing with <Request><DataMsg>----- \n");
253: memset(dat.msg, 0, sizeof(dat.msg));
254: DataMsgSetAll(dat, Manipulators[RCMT], DATAMSG);
255: if (!VerifyBufferData(dat.msg))
256: {
257:     printf("Buffer mismatch. \n");
258:     return 0;
259: }
260: if ( !DataMsgGetAll(dat, Manipulators[RCMT], DATAMSG) ||
261:     !DataMsgSetAllInvalid(dat, Manipulators[RCMTI], DATAMSG) ||
262:     !DataMsgGetAllInvalid(dat, Manipulators[RCMTI], DATAMSG) )
263:     return 0;
264: printf("-----<Request><DataMsg> test done----- \n");
265:
266: printf("-----Testing with <Request><FaultMsg>----- \n");
267: memset(dat.msg, 0, sizeof(dat.msg));
268: DataMsgSetAll(dat, Manipulators[RCMT], FAULTMSG);
269: if (!VerifyBufferDataFault(dat.msg))
270: {
271:     printf("Buffer mismatch. \n");
272:     return 0;
273: }
274: if ( !DataMsgGetAll(dat, Manipulators[RCMT], FAULTMSG) ||
275:     !DataMsgSetAllInvalid(dat, Manipulators[RCMTI], FAULTMSG) ||
276:     !DataMsgGetAllInvalid(dat, Manipulators[RCMTI], FAULTMSG) )
277:     return 0;
278: printf("-----<Request><FaultMsg> test done----- \n");
279: //
280: //</Request>
281: //////////////////////////////////////
282:     return 0;
283:
284: }

```

```

285:
286: bool VerifyBufferData(char* Buffer)
287: {
288:     char CompareBuffer[128];
289:
290:     PUT_UCHAR(CompareBuffer, UCHAR_VAL);
291:     PUT_CHAR(&CompareBuffer[1], CHAR_VAL);
292:     PUT_USHORT(CompareBuffer+2, USHORT_VAL);
293:     PUT_SHORT(CompareBuffer+4, SHORT_VAL);
294:     PUT_ULONG(CompareBuffer+6, UINT_VAL);
295:     PUT_LONG(CompareBuffer+10, INT_VAL);
296:     PUT_FLOAT(CompareBuffer+14, FLOAT_VAL);
297:     PUT_DOUBLE(CompareBuffer+18, DOUBLE_VAL);
298:     for (int i = 0; i < 26; i++)
299:         if (Buffer[i] != CompareBuffer[i])
300:             return false;
301:     return true;
302: }
303:
304: bool VerifyBufferDataFault(char* Buffer)
305: {
306:     char CompareBuffer[128];
307:
308:     PUT_DOUBLE(CompareBuffer, DOUBLE_VAL);
309:     PUT_FLOAT(CompareBuffer+8, FLOAT_VAL);
310:     PUT_LONG(CompareBuffer+12, INT_VAL);
311:     PUT_ULONG(CompareBuffer+16, UINT_VAL);
312:     PUT_SHORT(CompareBuffer+20, SHORT_VAL);
313:     PUT_USHORT(CompareBuffer+22, USHORT_VAL);
314:     PUT_CHAR(&CompareBuffer[24], CHAR_VAL);
315:     PUT_UCHAR(&CompareBuffer[25], UCHAR_VAL);
316:     for (int i = 0; i < 26; i++)
317:         if (Buffer[i] != CompareBuffer[i])
318:             return false;
319:     return true;
320: }
321:
322: bool CmdMsgSetAll (SDMCommand &cmd, MessageManipulator& mm)
323: {
324:     printf("Testing setValue with SDMCommand - UINT08...");
325:     if (mm.setValue("VarUChar", cmd, UCHAR_VAL))

```

```

326:     printf("passed. \n");
327: else
328: {
329:     printf("failed. \n"); return false;
330: }
331:
332: printf("Testing setValue with SDMCommand - INT08...");
333: if (mm.setValue("VarChar", cmd, CHAR_VAL))
334:     printf("passed. \n");
335: else
336: {
337:     printf("failed. \n"); return false;
338: }
339:
340: printf("Testing setValue with SDMCommand - UINT16...");
341: if (mm.setValue("VarUShort", cmd, USHORT_VAL))
342:     printf("passed. \n");
343: else
344: {
345:     printf("failed. \n"); return false;
346: }
347:
348: printf("Testing setValue with SDMCommand - INT16...");
349: if (mm.setValue("VarShort", cmd, SHORT_VAL))
350:     printf("passed. \n");
351: else
352: {
353:     printf("failed. \n"); return false;
354: }
355:
356: printf("Testing setValue with SDMCommand - UINT32...");
357: if (mm.setValue("VarUInt", cmd, UINT_VAL))
358:     printf("passed. \n");
359: else
360: {
361:     printf("failed. \n"); return false;
362: }
363:
364: printf("Testing setValue with SDMCommand - INT32...");
365: if (mm.setValue("VarInt", cmd, INT_VAL))
366:     printf("passed. \n");

```

```

367:     else
368:     {
369:         printf("failed. \n"); return false;
370:     }
371:
372:     printf("Testing setValue with SDMCommand - FLOAT32...");
373:     if (mm.setValue("VarFloat", cmd, FLOAT_VAL))
374:         printf("passed. \n");
375:     else
376:     {
377:         printf("failed. \n"); return false;
378:     }
379:
380:     printf("Testing setValue with SDMCommand - FLOAT64...");
381:     if (mm.setValue("VarDouble", cmd, DOUBLE_VAL))
382:         printf("passed. \n");
383:     else
384:     {
385:         printf("failed. \n"); return false;
386:     }
387:     return true;
388: }
389:
390: bool CmdMsgGetAll (SDMCommand& cmd, MessageManipulator& mm)
391: {
392:     if (mm.isMsgValid(cmd))
393:         printf("isMsgValid test passed. \n");
394:     else
395:         printf("isMsgValid test failed. \n");
396:
397:     printf("Testing getUINT08Value() with SDMCommand...");
398:     if (mm.getUINT08Value ("VarUChar", cmd) == UCHAR_VAL)
399:         printf("passed. \n");
400:     else
401:     {
402:         printf("failed. \n"); return false;
403:     }
404:
405:     bool Valid;
406:     printf("Testing getUINT08Value() with SDMCommand with Valid returned...");
407:     if (mm.getUINT08Value ("VarUChar", cmd, &Valid) == UCHAR_VAL && Valid)

```

```

408:     printf("passed. \n");
409: else
410: {
411:     printf("failed. \n"); return false;
412: }
413:
414: printf("Testing getINT08Value() with SDMCommand...");
415: if (mm.getINT08Value ("VarChar", cmd) == CHAR_VAL)
416:     printf("passed. \n");
417: else
418: {
419:     printf("failed. \n"); return false;
420: }
421:
422: printf("Testing getUINT16Value() with SDMCommand...");
423: if (mm.getUINT16Value ("VarUShort", cmd) == USHORT_VAL)
424:     printf("passed. \n");
425: else
426: {
427:     printf("failed. \n"); return false;
428: }
429:
430: printf("Testing getINT16Value() with SDMCommand...");
431: if (mm.getINT16Value ("VarShort", cmd) == SHORT_VAL)
432:     printf("passed. \n");
433: else
434: {
435:     printf("failed. \n"); return false;
436: }
437:
438: printf("Testing getUINT32Value() with SDMCommand...");
439: if (mm.getUINT32Value ("VarUInt", cmd) == UINT_VAL)
440:     printf("passed. \n");
441: else
442: {
443:     printf("failed. \n"); return false;
444: }
445:
446: printf("Testing getINT32Value() with SDMCommand...");
447: if (mm.getINT32Value ("VarInt", cmd) == INT_VAL)
448:     printf("passed. \n");

```

```

449:     else
450:     {
451:         printf("failed. \n"); return false;
452:     }
453:
454:     printf("Testing getFLOAT32Value() with SDMCommand...");
455:     if (mm.getFLOAT32Value ("VarFloat", cmd) == FLOAT_VAL)
456:         printf("passed. \n");
457:     else
458:     {
459:         printf("failed. \n"); return false;
460:     }
461:
462:     printf("Testing getFLOAT64Value() with SDMCommand...");
463:     if (mm.getFLOAT64Value ("VarDouble", cmd) == DOUBLE_VAL)
464:         printf("passed. \n");
465:     else
466:     {
467:         printf("failed. \n"); return false;
468:     }
469:     return true;
470: }
471:
472: bool CmdMsgSetAllInvalid (SDMCommand &cmd, MessageManipulator& mm)
473: {
474:     if (!mm.setValue("VarUChar", cmd, UCHAR_INVALID) ||
475:         !mm.setValue("VarChar", cmd, CHAR_INVALID) ||
476:         !mm.setValue("VarUShort", cmd, USHORT_INVALID) ||
477:         !mm.setValue("VarShort", cmd, SHORT_INVALID) ||
478:         !mm.setValue("VarUInt", cmd, UINT_INVALID) ||
479:         !mm.setValue("VarInt", cmd, INT_INVALID) ||
480:         !mm.setValue("VarFloat", cmd, FLOAT_INVALID) ||
481:         !mm.setValue("VarDouble", cmd, DOUBLE_INVALID) )
482:     {
483:         printf("CmdMsgSetAllInvalid failed. \n");
484:         return false;
485:     }
486:     return true;
487: }
488:
489: /*

```

```

490:   Returns all of the invalid data, the Valid flag should be set to false for each item.
491: */
492: bool CmdMsgGetAllInvalid (SDMCommand& cmd, MessageManipulator& mm)
493: {
494:     bool Valid;
495:
496:     if (!mm.isMsgValid(cmd))
497:         printf("isMsgValid test passed. \n");
498:     else
499:     {
500:         printf("isMsgValid test failed. \n"); return false;
501:     }
502:
503:     printf("Testing getUINT08Value() with SDMCommand invalid data...");
504:     if (mm.getUINT08Value ("VarUChar", cmd, &Valid) == UCHAR_INVALID && !Valid)
505:         printf("passed. \n");
506:     else
507:     {
508:         printf("failed. \n"); return false;
509:     }
510:
511:     printf("Testing getINT08Value() with SDMCommand invalid data...");
512:     if (mm.getINT08Value ("VarChar", cmd, &Valid) == CHAR_INVALID && !Valid)
513:         printf("passed. \n");
514:     else
515:     {
516:         printf("failed. \n"); return false;
517:     }
518:
519:     printf("Testing getUINT16Value() with SDMCommand invalid data...");
520:     if (mm.getUINT16Value ("VarUShort", cmd, &Valid) == USHORT_INVALID && !Valid)
521:         printf("passed. \n");
522:     else
523:     {
524:         printf("failed. \n"); return false;
525:     }
526:
527:     printf("Testing getINT16Value() with SDMCommand invalid data...");
528:     if (mm.getINT16Value ("VarShort", cmd, &Valid) == SHORT_INVALID && !Valid)
529:         printf("passed. \n");
530:     else

```



```

531:  {
532:      printf("failed. \n"); return false;
533:  }
534:
535:  printf("Testing getUINT32Value() with SDMCommand invalid data...");
536:  if (mm.getUINT32Value ("VarUInt", cmd, &Valid) == UINT_INVALID && !Valid)
537:      printf("passed. \n");
538:  else
539:  {
540:      printf("failed. \n"); return false;
541:  }
542:
543:  printf("Testing getINT32Value() with SDMCommand invalid data...");
544:  if (mm.getINT32Value ("VarInt", cmd, &Valid) == INT_INVALID && !Valid)
545:      printf("passed. \n");
546:  else
547:  {
548:      printf("failed. \n"); return false;
549:  }
550:
551:  printf("Testing getFLOAT32Value() with SDMCommand invalid data...");
552:  if (mm.getFLOAT32Value ("VarFloat", cmd, &Valid) == FLOAT_INVALID && !Valid)
553:      printf("passed. \n");
554:  else
555:  {
556:      printf("failed. \n"); return false;
557:  }
558:
559:  printf("Testing getFLOAT64Value() with SDMCommand invalid data...");
560:  if (mm.getFLOAT64Value ("VarDouble", cmd, &Valid) == DOUBLE_INVALID && !Valid)
561:      printf("passed. \n");
562:  else
563:  {
564:      printf("failed. \n"); return false;
565:  }
566:  return true;
567: }
568:
569: bool CmdMsgSetArray(SDMCommand& cmd, MessageManipulator& mm)
570: {
571:     char Array[64];

```

```

572:   char *RecvArray;
573:   for (unsigned int i = 0; i < sizeof(Array); i++)
574:       Array[i] = i;
575:
576:   printf("Testing setArray with SDMCommand...");
577:   if (!mm.setArray("VarArray",cmd,Array,sizeof(Array)))
578:   {
579:       printf("failed. \n");
580:       return false;
581:   }
582:   printf("passed. \n");
583:
584:   int Length;
585:   printf("Testing getArray with SDMCommand...");
586:
587:   RecvArray = (char*)mm.getArray("VarArray",cmd,Length);
588:   if (RecvArray == NULL)
589:   {
590:       printf("failed. \n");
591:       return false;
592:   }
593:
594:   for (int i = 0; i < Length; i++)
595:   {
596:       if (RecvArray[i] != Array[i])
597:       {
598:           printf("Buffer mismatch. \n");
599:           return false;
600:       }
601:   }
602:   printf("passed. \n");
603:   return true;
604: }
605:
606:
607: bool DataMsgSetAll (SDMData &dat, MessageManipulator& mm, enum msg_type type)
608: {
609:     printf("Testing setValue with SDMData - UINT08...");
610:     if (mm.setValue("VarUChar", dat, UCHAR_VAL, type))
611:         printf("passed. \n");
612:     else

```

```

613:  {
614:      printf("failed. \n"); return false;
615:  }
616:
617:  printf("Testing setValue with SDMData - INT08...");
618:  if (mm.setValue("VarChar", dat, CHAR_VAL, type))
619:      printf("passed. \n");
620:  else
621:  {
622:      printf("failed. \n"); return false;
623:  }
624:
625:  printf("Testing setValue with SDMData - UINT16...");
626:  if (mm.setValue("VarUShort", dat, USHORT_VAL, type))
627:      printf("passed. \n");
628:  else
629:  {
630:      printf("failed. \n"); return false;
631:  }
632:
633:  printf("Testing setValue with SDMData - INT16...");
634:  if (mm.setValue("VarShort", dat, SHORT_VAL, type))
635:      printf("passed. \n");
636:  else
637:  {
638:      printf("failed. \n"); return false;
639:  }
640:
641:  printf("Testing setValue with SDMData - UINT32...");
642:  if (mm.setValue("VarUInt", dat, UINT_VAL, type))
643:      printf("passed. \n");
644:  else
645:  {
646:      printf("failed. \n"); return false;
647:  }
648:
649:  printf("Testing setValue with SDMData - INT32...");
650:  if (mm.setValue("VarInt", dat, INT_VAL, type))
651:      printf("passed. \n");
652:  else
653:  {

```

```

654:     printf("failed. \n"); return false;
655: }
656:
657: printf("Testing setValue with SDMData - FLOAT32...");
658: if (mm.setValue("VarFloat", dat, FLOAT_VAL, type))
659:     printf("passed. \n");
660: else
661: {
662:     printf("failed. \n"); return false;
663: }
664:
665: printf("Testing setValue with SDMData - FLOAT64...");
666: if (mm.setValue("VarDouble", dat, DOUBLE_VAL, type))
667:     printf("passed. \n");
668: else
669: {
670:     printf("failed. \n"); return false;
671: }
672: return true;
673: }
674:
675: bool DataMsgSetAllInvalid (SDMData &dat, MessageManipulator& mm, enum msg_type type)
676: {
677:     if ( !mm.setValue("VarUChar", dat, UCHAR_INVALID, type) ||
678:         !mm.setValue("VarChar", dat, CHAR_INVALID, type) ||
679:         !mm.setValue("VarUShort", dat, USHORT_INVALID, type) ||
680:         !mm.setValue("VarShort", dat, SHORT_INVALID, type) ||
681:         !mm.setValue("VarUInt", dat, UINT_INVALID, type) ||
682:         !mm.setValue("VarInt", dat, INT_INVALID, type) ||
683:         !mm.setValue("VarFloat", dat, FLOAT_INVALID, type) ||
684:         !mm.setValue("VarDouble", dat, DOUBLE_INVALID, type) )
685:     {
686:         printf("DataMsgSetAllInvalid failed. \n");
687:         return false;
688:     }
689:     printf("DataMsgSetAllInvalid passed. \n");
690:     return true;
691: }
692:
693: bool DataMsgGetAll (SDMData& dat, MessageManipulator& mm, enum msg_type type)
694: {

```

```

695:   if (mm.isMsgValid(dat, type))
696:       printf("isMsgValid test passed. \n");
697:   else
698:       printf("isMsgValid test failed. \n");
699:
700:   printf("Testing getUINT08Value() with SDMDData...");
701:   if (mm.getUINT08Value ("VarUChar", dat, type) == UCHAR_VAL)
702:       printf("passed. \n");
703:   else
704:   {
705:       printf("failed. \n"); return false;
706:   }
707:
708:   printf("Testing getINT08Value() with SDMDData...");
709:   if (mm.getINT08Value ("VarChar", dat, type) == CHAR_VAL)
710:       printf("passed. \n");
711:   else
712:   {
713:       printf("failed. \n"); return false;
714:   }
715:
716:   printf("Testing getUINT16Value() with SDMDData...");
717:   if (mm.getUINT16Value ("VarUShort", dat, type) == USHORT_VAL)
718:       printf("passed. \n");
719:   else
720:   {
721:       printf("failed. \n"); return false;
722:   }
723:
724:   printf("Testing getINT16Value() with SDMDData...");
725:   if (mm.getINT16Value ("VarShort", dat, type) == SHORT_VAL)
726:       printf("passed. \n");
727:   else
728:   {
729:       printf("failed. \n"); return false;
730:   }
731:
732:   printf("Testing getUINT32Value() with SDMDData...");
733:   if (mm.getUINT32Value ("VarUInt", dat, type) == UINT_VAL)
734:       printf("passed. \n");
735:   else

```

```

736:  {
737:      printf("failed. \n"); return false;
738:  }
739:
740:  printf("Testing getINT32Value() with SDMDData...");
741:  if (mm.getINT32Value ("VarInt", dat, type) == INT_VAL)
742:      printf("passed. \n");
743:  else
744:  {
745:      printf("failed. \n"); return false;
746:  }
747:
748:  printf("Testing getFLOAT32Value() with SDMDData...");
749:  if (mm.getFLOAT32Value ("VarFloat", dat, type) == FLOAT_VAL)
750:      printf("passed. \n");
751:  else
752:  {
753:      printf("failed. \n"); return false;
754:  }
755:
756:  printf("Testing getFLOAT64Value() with SDMDData...");
757:  if (mm.getFLOAT64Value ("VarDouble", dat, type) == DOUBLE_VAL)
758:      printf("passed. \n");
759:  else
760:  {
761:      printf("failed. \n"); return false;
762:  }
763:  return true;
764: }
765:
766: bool DataMsgGetAllInvalid (SDMDData& dat, MessageManipulator& mm, enum msg_type type)
767: {
768:     bool Valid;
769:
770:     if (!mm.isMsgValid(dat, type))
771:         printf("isMsgValid test passed. \n");
772:     else
773:     {
774:         printf("isMsgValid test failed. \n"); return false;
775:     }
776:

```

```

777: printf("Testing getUINT08Value() with SDMDData invalid data...");
778: if (mm.getUINT08Value ("VarUChar", dat, type, &Valid) == UCHAR_INVALID && !Valid)
779:     printf("passed. \n");
780: else
781: {
782:     printf("failed. \n"); return false;
783: }
784:
785: printf("Testing getINT08Value() with SDMDData invalid data...");
786: if (mm.getINT08Value ("VarChar", dat, type, &Valid) == CHAR_INVALID && !Valid)
787:     printf("passed. \n");
788: else
789: {
790:     printf("failed. \n"); return false;
791: }
792:
793: printf("Testing getUINT16Value() with SDMDData invalid data...");
794: if (mm.getUINT16Value ("VarUShort", dat, type, &Valid) == USHORT_INVALID && !Valid)
795:     printf("passed. \n");
796: else
797: {
798:     printf("failed. \n"); return false;
799: }
800:
801: printf("Testing getINT16Value() with SDMDData invalid data...");
802: if (mm.getINT16Value ("VarShort", dat, type, &Valid) == SHORT_INVALID && !Valid)
803:     printf("passed. \n");
804: else
805: {
806:     printf("failed. \n"); return false;
807: }
808:
809: printf("Testing getUINT32Value() with SDMDData invalid data...");
810: if (mm.getUINT32Value ("VarUInt", dat, type, &Valid) == UINT_INVALID && !Valid)
811:     printf("passed. \n");
812: else
813: {
814:     printf("failed. \n"); return false;
815: }
816:
817: printf("Testing getINT32Value() with SDMDData invalid data...");

```

```

818:   if (mm.getINT32Value ("VarInt", dat, type, &Valid) == INT_INVALID && !Valid)
819:       printf("passed. \n");
820:   else
821:   {
822:       printf("failed. \n"); return false;
823:   }
824:
825:   printf("Testing getFLOAT32Value() with SDMDData invalid data...");
826:   if (mm.getFLOAT32Value ("VarFloat", dat, type, &Valid) == FLOAT_INVALID && !Valid)
827:       printf("passed. \n");
828:   else
829:   {
830:       printf("failed. \n"); return false;
831:   }
832:
833:   printf("Testing getFLOAT64Value() with SDMDData invalid data...");
834:   if (mm.getFLOAT64Value ("VarDouble", dat, type, &Valid) == DOUBLE_INVALID &&
!Valid)
835:       printf("passed. \n");
836:   else
837:   {
838:       printf("failed. \n"); return false;
839:   }
840:   return true;
841: }
842:
843: bool SerMsgSetAll (SDMService &ser, MessageManipulator& mm)
844: {
845:   printf("Testing setValue with SDMService - UINT08...");
846:   if (mm.setValue("VarUChar", ser, UCHAR_VAL))
847:       printf("passed. \n");
848:   else
849:   {
850:       printf("failed. \n"); return false;
851:   }
852:
853:   printf("Testing setValue with SDMService - INT08...");
854:   if (mm.setValue("VarChar", ser, CHAR_VAL))
855:       printf("passed. \n");
856:   else
857:   {

```



```

858:     printf("failed. \n"); return false;
859: }
860:
861: printf("Testing setValue with SDMSERVICE - UINT16...");
862: if (mm.setValue("VarUShort", ser, USHORT_VAL))
863:     printf("passed. \n");
864: else
865: {
866:     printf("failed. \n"); return false;
867: }
868:
869: printf("Testing setValue with SDMSERVICE - INT16...");
870: if (mm.setValue("VarShort", ser, SHORT_VAL))
871:     printf("passed. \n");
872: else
873: {
874:     printf("failed. \n"); return false;
875: }
876:
877: printf("Testing setValue with SDMSERVICE - UINT32...");
878: if (mm.setValue("VarUInt", ser, UINT_VAL))
879:     printf("passed. \n");
880: else
881: {
882:     printf("failed. \n"); return false;
883: }
884:
885: printf("Testing setValue with SDMSERVICE - INT32...");
886: if (mm.setValue("VarInt", ser, INT_VAL))
887:     printf("passed. \n");
888: else
889: {
890:     printf("failed. \n"); return false;
891: }
892:
893: printf("Testing setValue with SDMSERVICE - FLOAT32...");
894: if (mm.setValue("VarFloat", ser, FLOAT_VAL))
895:     printf("passed. \n");
896: else
897: {
898:     printf("failed. \n"); return false;

```

```

899:     }
900:
901:     printf("Testing setValue with SDMSERVICE - FLOAT64...");
902:     if (mm.setValue("VarDouble", ser, DOUBLE_VAL))
903:         printf("passed. \n");
904:     else
905:     {
906:         printf("failed. \n"); return false;
907:     }
908:     return true;
909: }
910:
911: bool SerMsgSetAllInvalid (SDMSERVICE &ser, MessageManipulator& mm)
912: {
913:     if (!mm.setValue("VarUChar", ser, UCHAR_INVALID) ||
914:         !mm.setValue("VarChar", ser, CHAR_INVALID) ||
915:         !mm.setValue("VarUShort", ser, USHORT_INVALID) ||
916:         !mm.setValue("VarShort", ser, SHORT_INVALID) ||
917:         !mm.setValue("VarUInt", ser, UINT_INVALID) ||
918:         !mm.setValue("VarInt", ser, INT_INVALID) ||
919:         !mm.setValue("VarFloat", ser, FLOAT_INVALID) ||
920:         !mm.setValue("VarDouble", ser, DOUBLE_INVALID) )
921:     {
922:         printf("SerMsgSetAllInvalid failed. \n");
923:         return false;
924:     }
925:     return true;
926: }
927:
928: bool SerMsgGetAll (SDMSerreqst& ser, MessageManipulator& mm)
929: {
930:     if (mm.isMsgValid(ser))
931:         printf("isMsgValid test passed. \n");
932:     else
933:         printf("isMsgValid test failed. \n");
934:
935:     printf("Testing getUINT08Value() with SDMSerreqst...");
936:     if (mm.getUINT08Value ("VarUChar", ser) == UCHAR_VAL)
937:         printf("passed. \n");
938:     else
939:     {

```

```

940:     printf("failed. \n"); return false;
941: }
942:
943: printf("Testing getINT08Value() with SDMSerreqst...");
944: if (mm.getINT08Value ("VarChar", ser) == CHAR_VAL)
945:     printf("passed. \n");
946: else
947: {
948:     printf("failed. \n"); return false;
949: }
950:
951: printf("Testing getUINT16Value() with SDMSerreqst...");
952: if (mm.getUINT16Value ("VarUShort", ser) == USHORT_VAL)
953:     printf("passed. \n");
954: else
955: {
956:     printf("failed. \n"); return false;
957: }
958:
959: printf("Testing getINT16Value() with SDMSerreqst...");
960: if (mm.getINT16Value ("VarShort", ser) == SHORT_VAL)
961:     printf("passed. \n");
962: else
963: {
964:     printf("failed. \n"); return false;
965: }
966:
967: printf("Testing getUINT32Value() with SDMSerreqst...");
968: if (mm.getUINT32Value ("VarUInt", ser) == UINT_VAL)
969:     printf("passed. \n");
970: else
971: {
972:     printf("failed. \n"); return false;
973: }
974:
975: printf("Testing getINT32Value() with SDMSerreqst...");
976: if (mm.getINT32Value ("VarInt", ser) == INT_VAL)
977:     printf("passed. \n");
978: else
979: {
980:     printf("failed. \n"); return false;

```

```

981:  }
982:
983:  printf("Testing getFLOAT32Value() with SDMSerreqst...");
984:  if (mm.getFLOAT32Value ("VarFloat", ser) == FLOAT_VAL)
985:      printf("passed. \n");
986:  else
987:  {
988:      printf("failed. \n"); return false;
989:  }
990:
991:  printf("Testing getFLOAT64Value() with SDMSerreqst...");
992:  if (mm.getFLOAT64Value ("VarDouble", ser) == DOUBLE_VAL)
993:      printf("passed. \n");
994:  else
995:  {
996:      printf("failed. \n"); return false;
997:  }
998:  return true;
999: }
1000:
1001: bool SerMsgGetAllInvalid (SDMSerreqst& ser, MessageManipulator& mm)
1002: {
1003:  bool Valid;
1004:
1005:  if (!mm.isMsgValid(ser))
1006:      printf("isMsgValid test passed. \n");
1007:  else
1008:  {
1009:      printf("isMsgValid test failed. \n"); return false;
1010:  }
1011:
1012:  printf("Testing getUINT08Value() with SDMSerreqst invalid data...");
1013:  if (mm.getUINT08Value ("VarUChar", ser, &Valid) == UCHAR_INVALID && !Valid)
1014:      printf("passed. \n");
1015:  else
1016:  {
1017:      printf("failed. \n"); return false;
1018:  }
1019:
1020:  printf("Testing getINT08Value() with SDMSerreqst invalid data...");
1021:  if (mm.getINT08Value ("VarChar", ser, &Valid) == CHAR_INVALID && !Valid)

```

```

1022:     printf("passed. \n");
1023: else
1024: {
1025:     printf("failed. \n"); return false;
1026: }
1027:
1028: printf("Testing getUINT16Value() with SDMSerreqst invalid data...");
1029: if (mm.getUINT16Value ("VarUShort", ser, &Valid) == USHORT_INVALID && !Valid)
1030:     printf("passed. \n");
1031: else
1032: {
1033:     printf("failed. \n"); return false;
1034: }
1035:
1036: printf("Testing getINT16Value() with SDMSerreqst invalid data...");
1037: if (mm.getINT16Value ("VarShort", ser, &Valid) == SHORT_INVALID && !Valid)
1038:     printf("passed. \n");
1039: else
1040: {
1041:     printf("failed. \n"); return false;
1042: }
1043:
1044: printf("Testing getUINT32Value() with SDMSerreqst invalid data...");
1045: if (mm.getUINT32Value ("VarUInt", ser, &Valid) == UINT_INVALID && !Valid)
1046:     printf("passed. \n");
1047: else
1048: {
1049:     printf("failed. \n"); return false;
1050: }
1051:
1052: printf("Testing getINT32Value() with SDMSerreqst invalid data...");
1053: if (mm.getINT32Value ("VarInt", ser, &Valid) == INT_INVALID && !Valid)
1054:     printf("passed. \n");
1055: else
1056: {
1057:     printf("failed. \n"); return false;
1058: }
1059:
1060: printf("Testing getFLOAT32Value() with SDMSerreqst invalid data...");
1061: if (mm.getFLOAT32Value ("VarFloat", ser, &Valid) == FLOAT_INVALID && !Valid)
1062:     printf("passed. \n");

```

```
1063: else
1064: {
1065:     printf("failed. \n"); return false;
1066: }
1067:
1068: printf("Testing getFLOAT64Value() with SDMSerreqst invalid data...");
1069: if (mm.getFloat64Value ("VarDouble", ser, &Valid) == DOUBLE_INVALID && !Valid)
1070:     printf("passed. \n");
1071: else
1072: {
1073:     printf("failed. \n"); return false;
1074: }
1075: return true;
1076: }
1077:
```

File: sdm/app/test/ClassTests/SDMVarTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "../common/message/SDMVarReq.h"
4: #include "../common/message/SDMVarInfo.h"
5:
6: int main (int argc, char ** argv)
7: {
8:     SDMVarReq req;
9:     SDMVarInfo info;
10:    char MsgBuf[8096];
11:    char send_buf[BUFSIZE];
12:
13:    //Set request message info
14:    req.source.setPort(1020);
15:    req.destination.setPort(1234);
16:    req.msg_id.setInterface(11);
17:    req.id = 4;
18:    strcpy(req.variable, "TestVar");
19:    //Set info message info
20:    info.source.setPort(3443);
21:    info.id = 44;
22:    strcpy(info.var_xTEDS, "Test xTEDS");
23:
24:    printf("----SDMVarReq TEST---- \n");
25:    req.MsgToString(MsgBuf, sizeof(MsgBuf));
26:    printf ("Before any marshallng: ");
27:    printf("%s \n",MsgBuf);
28:    req.Marshal(send_buf);
29:    req.Unmarshal(send_buf);
30:    printf ("After unmarshalling: ");
31:    req.MsgToString(MsgBuf, sizeof(MsgBuf));
32:    printf("%s \n", MsgBuf);
33:
34:    printf(" \n \n----SDMVarInfo TEST---- \n");
35:    info.MsgToString(MsgBuf, sizeof(MsgBuf));
36:    printf ("Before any marshallng: ");
37:    printf("%s \n",MsgBuf);
38:    info.Marshal(send_buf);
39:    info.Unmarshal(send_buf);
```

```
40: printf ("After unmarshalling: ");
41: info.MsgToString(MsgBuf, sizeof(MsgBuf));
42: printf("%s \n", MsgBuf);
43:
44: printf(" \n \nFinished. \n");
45: return 0;
46: }
```


File: sdm/app/test/ClassTests/Makefile.uclinux

```
1: ifndef PETALINUX
2: $(error You must source the petalinux/settings.sh script before working with PetaLinux)
3: endif
4:
5: # Point to default PetaLinux root directory
6: ifndef ROOTDIR
7: ROOTDIR=$(PETALINUX)/software/petalinux-dist
8: endif
9:
10: PATH:=$(PATH):$(ROOTDIR)/tools
11:
12: UCLINUX_BUILD_USER = 1
13: -include $(ROOTDIR)/.config
14: -include $(ROOTDIR)/$(CONFIG_LINUXDIR)/.config
15: LIBCDIR = $(CONFIG_LIBCDIR)
16: -include $(ROOTDIR)/config.arch
17: ROMFSDIR=$(ROOTDIR)/romfs
18: ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh
19:
20: LDFLAGS+=-L../../common
21: LDLIBS+=-lpthread -lsdm
22:
23: APPS = TimeTest
24:
25: all: $(APPS)
26:
27: TimeTest: TimeTest.o
28: $(CXX) $(LDFLAGS) -o $@ $^ $(LDLIBS)
29:
30: clean:
31: -rm -f $(APPS) *.elf *.gdb *.o
32:
33: romfs:
34: $(ROMFSINST) TimeTest /bin/TimeTest
35:
36: %.o: %.cpp
37: $(CXX) -c $(CXXFLAGS) -o $@ $<
38:
39:
```

```
40: # Targets for the required .config files - if they don't exist, the tree isn't
41: # configured. Tell the user this, how to fix it, and exit.
42: ${ROOTDIR}/config.arch ${ROOTDIR}/.config:
43: @echo "Error: You must configure the PetaLinux tree before compiling your application"
44: @echo ""
45: @echo "Change directory to ../../petalinux-dist and 'make menuconfig' or 'make xconfig'"
46: @echo ""
47: @echo "Once the tree is configured, return to this directory, and re-run make."
48: @echo ""
49: @exit -1
50:
```

File: sdm/app/test/ClassTests/Makefile

```
1: include ../../Makefile.defs
2:
3:
4: .PHONY:    all clean distclean
5:
6:  BUILDTARGETS=MessageManipulatorTest  SDMMMessage_IDTest  SDMVarTest  SubManTest
TimeTest MessageTests ProviderSubscriptionListTest
7:
8: all: $(BUILDTARGETS)
9:
10: MessageManipulatorTest: MessageManipulatorTest.o xTEDSPoster.o
11: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
12:
13: SDMMMessage_IDTest: SDMMMessage_IDTest.o
14: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
15:
16: SDMVarTest: SDMVarTest.o
17: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
18:
19: SubManTest: SubManTest.o
20: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
21:
22: TimeTest: TimeTest.o
23: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
24:
25: MessageTests: MessageTests.o
26: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
27:
28: ProviderSubscriptionListTest: ProviderSubscriptionListTest.o ../../dm/ProviderSubscriptionList.o
../../dm/ProviderSubscription.o
29: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
30:
31: xTEDSPoster.o: ../DMTests/xTEDSPoster.cpp ../DMTests/xTEDSPoster.h
32: $(CXX) $(CXXFLAGS) -c $<
33:
34: %.o: %.cpp
35: $(CXX) $(CXXFLAGS) -c $<
36:
37: clean:
```

```
38: rm -f *.o
39:
40: distclean: clean
41: rm -f $(BUILDTARGETS) *~
```

File: sdm/app/test/ClassTests/MessageTests.cpp

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <unistd.h>
4: #include <pthread.h>
5: #include <sys/socket.h>
6: #include <netinet/in.h>
7: #include <arpa/inet.h>
8:
9: #include "../common/message/SDMData.h"
10: #include "../common/MessageManager/MessageManager.h"
11:
12:
13: const unsigned int MYPORT = 7453;
14: const unsigned int MYPORT2 = 9745;
15:
16: void* Sender(void*);
17: void* Forwarder(void*);
18: bool DataCheck(const SDMData& Message);
19:
20: int main (int argc, char** argv)
21: {
22:     pthread_t SenderThread, ForwarderThread;
23:
24:     if (0 != pthread_create(&SenderThread, NULL, Sender, NULL))
25:     {
26:         printf("Could not create sender thread. \n");
27:         return -1;
28:     }
29:
30:     if (0 != pthread_create(&ForwarderThread, NULL, Forwarder, NULL))
31:     {
32:         printf("Could not create forwarder thread. \n");
33:         return -1;
34:     }
35:
36:     pthread_join(SenderThread, NULL);
37:     pthread_join(ForwarderThread, NULL);
38:
39:     return 0;
```

```

40: }
41:
42: void* Sender(void* ignored)
43: {
44:     MessageManager mm;
45:     SDMDData dat;
46:     for (int i = 0; i < 256; i++)
47:         dat.msg[i] = 0x45;
48:     dat.length = 256;
49:     mm.Async_Init(MYPORT);
50:     char buf[BUFSIZE];
51:
52:     SDMComponent_ID Receiver;
53:     Receiver.setAddress(inet_addr("127.0.0.1"));
54:     Receiver.setPort(MYPORT2);
55:
56:     while(1)
57:     {
58:         if (mm.IsReady())
59:         {
60:             if (mm.GetMessage(buf) == SDM_Data)
61:             {
62:                 SDMDData received;
63:                 received.Unmarshal(buf);
64:
65:                 printf("(Sender)      Received  SDMDData  with  timestamp    %ld,    %ld.
\n",received.GetSecondsStamp(), received.GetSubSecondsStamp());
66:                 fflush(NULL);
67:                 if (!DataCheck(received))
68:                 {
69:                     printf("(Sender) ERROR message doesn't match! \n");
70:                 }
71:             }
72:             else
73:                 printf("(Sender) Unknown message received. \n");
74:
75:         }
76:         // Send the message
77:         printf("(Sender) Sending message... \n");
78:         dat.SendTo(Receiver);
79:

```

```

80:     sleep(1);
81: }
82:
83: return NULL;
84: }
85:
86: void* Forwarder(void* ignored)
87: {
88:     MessageManager mm;
89:     SDMData dat;
90:     mm.Async_Init(MYPORT2);
91:     char buf[BUFSIZE];
92:
93:     SDMComponent_ID Receiver;
94:     Receiver.setAddress(inet_addr("127.0.0.1"));
95:     Receiver.setPort(MYPORT);
96:
97:     while(1)
98:     {
99:         if (mm.IsReady())
100:         {
101:             if (mm.GetMessage(buf) == SDM_Data)
102:             {
103:                 SDMData received;
104:                 long Length = received.Unmarshal(buf);
105:
106:                 printf("(Forwarder) Received SDMData with timestamp %ld, %ld length is %ld.
\n",received.GetSecondsStamp(), received.GetSubSecondsStamp(), Length);
107:
108:
109:                 Length = received.Forward(Receiver);
110:                 printf("(Forwarder) Forwarding message length is %ld. \n", Length);
111:                 fflush(NULL);
112:             }
113:             else
114:                 printf("(Forwarder) Unknown message received. \n");
115:         }
116:         sleep(1);
117:     }
118:     return NULL;
119: }

```

```
120:
121: bool DataCheck(const SDMData& Message)
122: {
123:     for (int i = 0; i < 256; i++)
124:     {
125:         if (Message.msg[i] != 0x45)
126:             return false;
127:     }
128:     return true;
129: }
130:
131:
132:
```


File: sdm/app/test/ClassTests/SDMMessage_IDTest.cpp

```
1: #include <stdio.h>
2: #include "../common/message/SDMMessage_ID.h"
3:
4: int main (int argc, char ** argv)
5: {
6:     printf ("***STARTING TEST*** \n");
7:     SDMMessage_ID testId1(188, 4);
8:     SDMMessage_ID testId2(188, 4);
9:     SDMMessage_ID testId3;
10:
11:     printf("Id1.getInterface() = %d \n",testId1.getInterface());
12:     printf("Id1.getMessage() = %d \n",testId1.getMessage());
13:     if (testId1 == testId2)
14:         printf("operator== test passed. \n");
15:     else
16:     {
17:         printf("operator== test failed. \n");
18:         return 0;
19:     }
20:     testId3 = testId1;
21:     if (testId3 == testId1)
22:         printf("operator= test passed. \n");
23:     else
24:     {
25:         printf("operator= test failed. \n");
26:         return 0;
27:     }
28:     testId3.setInterface(1);
29:     if (testId3.getInterface() == 1)
30:         printf("set/getInterface() test passed. \n");
31:     else
32:     {
33:         printf("set/getInterface() test failed. \n");
34:         return 0;
35:     }
36:     testId3.setMessage(50);
37:     if (testId3.getMessage() == 50)
38:         printf("set/getMessage() test passed. \n");
39:     else
```

```

40: {
41:     printf("set/getMessage() test failed. \n");
42:     return 0;
43: }
44:
45: char buf[2];
46: int result = testId3.Marshal(buf,0);
47: if (buf[0] == 1 && buf[1] == 50 && result == 2)
48:     printf("marshal test passed. \n");
49: else
50: {
51:     printf("marshal test failed. \n");
52:     return 0;
53: }
54: testId2.Unmarshal(buf, 0);
55: if (testId2 == testId3)
56:     printf("unmarshal test passed. \n");
57: else
58: {
59:     printf("unmarshal test failed. \n");
60:     return 0;
61: }
62:
63: printf("****TEST COMPLETE**** \n");
64:
65: }

```

File: sdm/app/test/ClassTests/MessageManipulatorTest.xml

```
1: <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2:         <xTEDS                                version="11.16"                                name="Robo_Hub_xTEDS"
xmlns="http://www.interfacecontrol.com/SPA/xTEDS">
3:   <Device modelId="0001" kind="Robust_Hub" name="MessageManipulatorTest"/>
4:   <Interface id="1" name="TestInterface1">
5:     <Variable format="UINT08" kind="test" name="VarUChar"/>
6:   <Variable format="INT08" kind="test" name="VarChar"/>
7:   <Variable format="UINT16" kind="test" name="VarUShort"/>
8:   <Variable format="INT16" kind="test" name="VarShort"/>
9:   <Variable format="UINT32" kind="test" name="VarUInt"/>
10:  <Variable format="INT32" kind="test" name="VarInt"/>
11:  <Variable format="FLOAT32" kind="test" name="VarFloat"/>
12:  <Variable format="FLOAT64" kind="test" name="VarDouble"/>
13:    <Variable format="UINT08" kind="test" name="VarArray" length="64"/>
14:
15:    <Notification>
16:      <DataMsg msgRate="1" msgArrival="PERIODIC" id="1" name="NotDatMsgTest">
17:        <VariableRef name="VarUChar"/>
18:        <VariableRef name="VarChar"/>
19:        <VariableRef name="VarUShort"/>
20:        <VariableRef name="VarShort"/>
21:        <VariableRef name="VarUInt"/>
22:        <VariableRef name="VarInt"/>
23:        <VariableRef name="VarFloat"/>
24:        <VariableRef name="VarDouble"/>
25:        <VariableRef name="VarArray"/>
26:      </DataMsg>
27:    <FaultMsg id="23" name="NotFaultMsgTest">
28:      <VariableRef name="VarDouble"/>
29:      <VariableRef name="VarFloat"/>
30:      <VariableRef name="VarInt"/>
31:      <VariableRef name="VarUInt"/>
32:      <VariableRef name="VarShort"/>
33:      <VariableRef name="VarUShort"/>
34:      <VariableRef name="VarChar"/>
35:      <VariableRef name="VarUChar"/>
36:      <VariableRef name="VarArray"/>
37:    </FaultMsg>
38:  </Notification>
```

```

39:     <Command>
40:         <CommandMsg id="2" name="CmdCmdMsgTest">
41:             <VariableRef name="VarUChar"/>
42:             <VariableRef name="VarChar"/>
43:             <VariableRef name="VarUShort"/>
44:             <VariableRef name="VarShort"/>
45:             <VariableRef name="VarUInt"/>
46:             <VariableRef name="VarInt"/>
47:             <VariableRef name="VarFloat"/>
48:             <VariableRef name="VarDouble"/>
49:             <VariableRef name="VarArray"/>
50:         </CommandMsg>
51: <FaultMsg id="5" name="CmdFaultMsgTest">
52:     <VariableRef name="VarDouble"/>
53:     <VariableRef name="VarFloat"/>
54:     <VariableRef name="VarInt"/>
55:     <VariableRef name="VarUInt"/>
56:     <VariableRef name="VarShort"/>
57:     <VariableRef name="VarUShort"/>
58:     <VariableRef name="VarChar"/>
59:     <VariableRef name="VarUChar"/>
60:     <VariableRef name="VarArray"/>
61: </FaultMsg>
62: </Command>
63: <Request>
64:     <CommandMsg id="3" name="ReqCmdMsgTest">
65:         <VariableRef name="VarUChar"/>
66:         <VariableRef name="VarChar"/>
67:         <VariableRef name="VarUShort"/>
68:         <VariableRef name="VarShort"/>
69:         <VariableRef name="VarUInt"/>
70:         <VariableRef name="VarInt"/>
71:         <VariableRef name="VarFloat"/>
72:         <VariableRef name="VarDouble"/>
73:         <VariableRef name="VarArray"/>
74:     </CommandMsg>
75: <DataReplyMsg id="4" name="ReqDataRplyMsgTest">
76:     <VariableRef name="VarUChar"/>
77:     <VariableRef name="VarChar"/>
78:     <VariableRef name="VarUShort"/>
79:     <VariableRef name="VarShort"/>

```

```

80:     <VariableRef name="VarUInt"/>
81:     <VariableRef name="VarInt"/>
82:     <VariableRef name="VarFloat"/>
83:     <VariableRef name="VarDouble"/>
84:     <VariableRef name="VarArray"/>
85: </DataReplyMsg>
86: <FaultMsg id="5" name="ReqFaultMsgTest">
87:     <VariableRef name="VarDouble"/>
88:     <VariableRef name="VarFloat"/>
89:     <VariableRef name="VarInt"/>
90:     <VariableRef name="VarUInt"/>
91:     <VariableRef name="VarShort"/>
92:     <VariableRef name="VarUShort"/>
93:     <VariableRef name="VarChar"/>
94:     <VariableRef name="VarUChar"/>
95:     <VariableRef name="VarArray"/>
96: </FaultMsg>
97: </Request>
98: </Interface>
99: <Interface id="2" name="TestInterface2">
100:     <Variable format="UINT08" kind="test" name="VarUChar" invalidValue="13"/>
101:     <Variable format="INT08" kind="test" name="VarChar" invalidValue="13"/>
102:     <Variable format="UINT16" kind="test" name="VarUShort" invalidValue="13"/>
103:     <Variable format="INT16" kind="test" name="VarShort" invalidValue="13"/>
104:     <Variable format="UINT32" kind="test" name="VarUInt" invalidValue="13"/>
105:     <Variable format="INT32" kind="test" name="VarInt" invalidValue="13"/>
106:     <Variable format="FLOAT32" kind="test" name="VarFloat" invalidValue="13"/>
107:     <Variable format="FLOAT64" kind="test" name="VarDouble" invalidValue="13.0"/>
108:     <Variable format="UINT08" kind="test" name="VarArray" length="64"/>
109:
110:     <Notification>
111:         <DataMsg msgRate="1" msgArrival="PERIODIC" id="1" name="NotDatMsgTestInvalid">
112:             <VariableRef name="VarUChar"/>
113:             <VariableRef name="VarChar"/>
114:             <VariableRef name="VarUShort"/>
115:             <VariableRef name="VarShort"/>
116:             <VariableRef name="VarUInt"/>
117:             <VariableRef name="VarInt"/>
118:             <VariableRef name="VarFloat"/>
119:             <VariableRef name="VarDouble"/>
120:             <VariableRef name="VarArray"/>

```

```

121:     </DataMsg>
122: <FaultMsg id="23" name="NotFaultMsgTest">
123:     <VariableRef name="VarDouble"/>
124:     <VariableRef name="VarFloat"/>
125:     <VariableRef name="VarInt"/>
126:     <VariableRef name="VarUInt"/>
127:     <VariableRef name="VarShort"/>
128:     <VariableRef name="VarUShort"/>
129:     <VariableRef name="VarChar"/>
130:     <VariableRef name="VarUChar"/>
131:     <VariableRef name="VarArray"/>
132: </FaultMsg>
133: </Notification>
134: <Command>
135:     <CommandMsg id="2" name="CmdCmdMsgTestInvalid">
136:         <VariableRef name="VarUChar"/>
137:         <VariableRef name="VarChar"/>
138:         <VariableRef name="VarUShort"/>
139:         <VariableRef name="VarShort"/>
140:         <VariableRef name="VarUInt"/>
141:         <VariableRef name="VarInt"/>
142:         <VariableRef name="VarFloat"/>
143:         <VariableRef name="VarDouble"/>
144:         <VariableRef name="VarArray"/>
145:     </CommandMsg>
146: <FaultMsg id="5" name="CmdFaultMsgTestInvalid">
147:     <VariableRef name="VarDouble"/>
148:     <VariableRef name="VarFloat"/>
149:     <VariableRef name="VarInt"/>
150:     <VariableRef name="VarUInt"/>
151:     <VariableRef name="VarShort"/>
152:     <VariableRef name="VarUShort"/>
153:     <VariableRef name="VarChar"/>
154:     <VariableRef name="VarUChar"/>
155:     <VariableRef name="VarArray"/>
156: </FaultMsg>
157: </Command>
158: <Request>
159:     <CommandMsg id="3" name="ReqCmdMsgTestInvalid">
160:         <VariableRef name="VarUChar"/>
161:         <VariableRef name="VarChar"/>

```

```

162:     <VariableRef name="VarUShort"/>
163:     <VariableRef name="VarShort"/>
164:     <VariableRef name="VarUInt"/>
165:     <VariableRef name="VarInt"/>
166:     <VariableRef name="VarFloat"/>
167:     <VariableRef name="VarDouble"/>
168:     <VariableRef name="VarArray"/>
169:     </CommandMsg>
170: <DataReplyMsg id="4" name="ReqDataRplyMsgTestInvalid">
171:     <VariableRef name="VarUChar"/>
172:     <VariableRef name="VarChar"/>
173:     <VariableRef name="VarUShort"/>
174:     <VariableRef name="VarShort"/>
175:     <VariableRef name="VarUInt"/>
176:     <VariableRef name="VarInt"/>
177:     <VariableRef name="VarFloat"/>
178:     <VariableRef name="VarDouble"/>
179:     <VariableRef name="VarArray"/>
180: </DataReplyMsg>
181: <FaultMsg id="5" name="ReqFaultMsgTestInvalid">
182:     <VariableRef name="VarDouble"/>
183:     <VariableRef name="VarFloat"/>
184:     <VariableRef name="VarInt"/>
185:     <VariableRef name="VarUInt"/>
186:     <VariableRef name="VarShort"/>
187:     <VariableRef name="VarUShort"/>
188:     <VariableRef name="VarChar"/>
189:     <VariableRef name="VarUChar"/>
190:     <VariableRef name="VarArray"/>
191: </FaultMsg>
192: </Request>
193: </Interface>
194: </xTEDS>

```

File: sdm/app/test/ClassTests/TimeTest.cpp

```
1: #include <stdio.h>
2: #include <unistd.h>
3: #include <pthread.h>
4: #include <signal.h>
5: #include "../common/Time/SDMTime.h"
6:
7: const unsigned int TimeValsSec[] = { 1, 8, 7, 3 };
8: const unsigned int TimeValsUSec [] = { 0, 20000, 86994, 238821};
9: unsigned int TimeValIndex = 0;
10: pthread_mutex_t TimeValMutex = PTHREAD_MUTEX_INITIALIZER;
11:
12: void *ThreadTimer(void *args);
13:
14: void CallBack (int Timer)
15: {
16: printf("Callback called by timer %d \n",Timer);
17: }
18:
19: int main (int argc, char ** argv)
20: {
21: printf("-----SDM Time Test Started----- \n");
22: printf("SDM_GetTime() ");
23: unsigned int Sec, USec;
24: SDM_GetTime(&Sec, &USec);
25: printf("sec=%u, usec=%u passed. \n",Sec,USec);
26: printf("SDM_GetTime(BadInput) ");
27: SDM_GetTime(NULL, NULL);
28: printf("passed. \n");
29:
30: SDM_TimeInit();
31:
32: printf(" \n \n");
33: printf("SDMSleep(1, 500) "); fflush(NULL);
34: SDM_Sleep(1, 500);
35: printf("passed. \n");
36:
37: pthread_t t1, t2, t3, t4;
38: pthread_create(&t1, NULL, ThreadTimer, NULL);
39: pthread_create(&t2, NULL, ThreadTimer, NULL);
```



```

40: pthread_create(&t3, NULL, ThreadTimer, NULL);
41: pthread_create(&t4, NULL, ThreadTimer, NULL);
42:
43: pthread_join(t1, NULL);
44: pthread_join(t2, NULL);
45: pthread_join(t3, NULL);
46: pthread_join(t4, NULL);
47:
48: printf("-----SDM Time Test Finished----- \n");
49:
50: return 0;
51: }
52:
53: void *ThreadTimer(void *arg)
54: {
55: unsigned int Sec, USec, Index, t1, t2;
56: pthread_mutex_lock(&TimeValMutex);
57: Index = TimeValIndex++;
58: Sec = TimeValsSec[Index];
59: USec = TimeValsUSec[Index++];
60: pthread_mutex_unlock(&TimeValMutex);
61: int TimerID;
62: printf("%u - SDM_CreateTimer(%u seconds, %u useconds) \n", Index, Sec, USec);
63: if ((TimerID=SDM_CreateTimer(Sec, USec, &CallBack, true)) < 0)
64:     printf("failed. \n");
65:
66: bool Finished = false;
67: int count = 0;
68: while (!Finished)
69: {
70:     SDM_Sleep(1, 0);
71:     count++;
72:     SDM_ReadTimer(TimerID, &t1, &t2);
73:     printf("%u - Current Timer remaining is %u, %u \n", TimerID, t1, t2);
74:     if (count == 10)
75:     {
76:         printf("%u - Deleting Timer. \n", Index);
77:         SDM_DeleteTimer(TimerID);
78:         printf("%u - Creating Timer. \n", Index);
79:         if ((TimerID=SDM_CreateTimer(Sec, USec, &CallBack, true)) < 0)
80:             printf("failed. \n");

```

```
81:     }
82:     if (count == 20)
83:         Finished = true;
84: }
85: printf("%u done. \n",Index);
86: return NULL;
87: }
```

File: sdm/app/test/PMTests/AppFailSilentTestConsumer.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMHeartbeat.h"
7: #include "../common/MessageManipulator/MessageManipulator.h"
8: #include "../common/MessageManager/MessageManager.h"
9:
10: #include <string.h>
11: #include <unistd.h>
12: #include <stdio.h>
13: #include <sys/types.h>
14: #include <sys/wait.h>
15: #include <signal.h>
16:
17: #define DATA_PROVIDER    1
18: #define SERVICE_PROVIDER 2
19:
20: long my_port;
21: SDMComponent_ID data_provider;
22: SDMComponent_ID service_provider;
23: SDMMessage_ID data_msg(0,0);
24: SDMMessage_ID service_msg(0,0);
25:
26: MessageManipulator data_manipulator;
27: MessageManipulator service_manipulator;
28:
29: void DataHandler(SDMData& dat,long length)
30: {
31:     SDMService request;
32:     short int_value;
33:     float float_value;
34:     static float geo_average = 0;
35:
36:     if((dat.source == data_provider)&&(dat.msg_id == data_msg))
37:     {
38:         int_value = data_manipulator.getUINT16Value("data",dat,DATAMSG);
39:         printf(" received value is %d \n", int_value);
```

```

40: }
41: }
42:
43: void RegInfoHandler(SDMRegInfo& info)
44: {
45:     SDMConsume consume;
46:     SDMReqReg req_reg;
47:
48:     //Set the port we will be receiving on
49:     consume.destination.setPort(my_port);
50:     //copy the sensor id into the consume message
51:     consume.source=info.source;
52:     //copy the msg id into the consume message
53:     consume.msg_id=info.msg_id;
54:
55:     switch(info.id)
56:     {
57:     case DATA_PROVIDER:
58:         if(info.type == 1)    // cancellation
59:         {
60:             data_provider.setSensorID(0);
61:             printf("Data provider failed . . . ");
62:             printf("Searching for new provider \n");
63:             //request info on integer data providers,
64:             //the DM will repost software tasks that
65:             // could satisfy our requirements
66:
67:             //Set variable name
68:             strcpy(req_reg.item_name,"data");
69:             //Set the quallist can be empty
70:             strcpy(req_reg.quallist,"< format= \"UINT16 \"/>");
71:             req_reg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
72:             req_reg.destination.setPort(my_port);
73:             req_reg.id = DATA_PROVIDER;
74:             //req_reg.Send();
75:         }
76:     else
77:     {
78:         if(data_provider.getSensorID() == 0)
79:         {
80:             data_provider = info.source;

```

```

81:     data_msg = info.msg_id;
82:     data_manipulator.setMsgDef(info.msg_def);
83:     printf("New Data provider found");
84:     printf(" \n \n%s \n",info.msg_def);
85:     //Send the consume message
86:     consume.Send();
87: }
88: else
89: {
90:     printf("Data provider found");
91:     printf(" - not used \n \n%s \n",info.msg_def);
92: }
93: }
94: break;
95: }
96:
97: }
98:
99: int main(int argc,char** argv)
100: {
101:     MessageManager mm;
102:     SDMDData dat;
103:     SDMRegInfo info;
104:     SDMReqReg req_reg;
105:     char buf[BUFSIZE];
106:     long length;
107:
108:     //initialize consumer
109:     SDMInit(argc,argv);
110:     my_port = getPort();
111:     if(my_port == SDM_PM_NOT_AVAILABLE)
112:     {
113:         printf("No PM is available to get port from! \n");
114:         return 0;
115:     }
116:     mm.Async_Init(my_port);
117:
118:     printf("Consumer listening on port %ld \n",my_port);
119:
120:     while(1)
121:     {

```

```

122:     if (mm.IsReady())
123:     {
124:         SendHeartbeat();
125: #ifdef WIN32
126:         switch(mm.GetMsg(buf,length))
127: #else
128:         switch(mm.GetMessage(buf,length))
129: #endif
130:     {
131:         case SDM_Data:
132:             dat.Unmarshal(buf,length);
133:             DataHandler(dat,length);
134:             break;
135:         case SDM_RegInfo:
136:             if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)
137:             {
138:                 RegInfoHandler(info);
139:             }
140:             break;
141:         default:
142:             printf("Unexpected message \n");
143:     }
144: }
145: else
146: { //check for data and service providers
147:     if(data_provider.getSensorID() == 0)
148:     {
149:         //request info on integer data providers
150:         //Set variable name
151:         strcpy(req_reg.item_name,"data");
152:         //Set the quallist can be empty
153:         strcpy(req_reg.quallist,"< format= \"UINT16 \"/>");
154:         req_reg.reply
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
155:         req_reg.destination.setPort(my_port);
156:         req_reg.id = DATA_PROVIDER;
157:         req_reg.Send();
158:         printf("Searching for new data provider \n");
159:         sleep(2);
160:     }
161:     usleep(1000);

```

=

162: }
163: }
164: }

File: sdm/app/test/PMTests/Makefile

```
1: include ../../Makefile.defs
2:
3: .PHONY: all clean distclean
4:
5: BUILD_TARGETS = AppFailSilentTestProducer AppFailSilentTestConsumer AppFailSilentTest
6:
7: all: $(BUILD_TARGETS)
8:
9: AppFailSilentTestProducer: AppFailSilentTestProducer.cpp
10: $(CXX) $(CXXFLAGS) -o $@ $< -L../../common/ -lSDM -lpthread
11:
12: AppFailSilentTestConsumer: AppFailSilentTestConsumer.cpp
13: $(CXX) $(CXXFLAGS) -o $@ $< -L../../common/ -lSDM -lpthread
14:
15: AppFailSilentTest: AppFailSilentTest.cpp
16: $(CXX) $(CXXFLAGS) -o $@ $< -L../../common/ -lSDM -lpthread
17:
18: clean:
19: rm -f $(BUILD_TARGETS)
20:
21: distclean: clean
```


File: sdm/app/test/PMTests/AppFailSilentTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5:
6: #include "../common/MessageManager/MessageManager.h"
7: #include "../common/message/SDMReqReg.h"
8: #include "../common/message/SDMRegInfo.h"
9: #include "../common/message/SDMCommand.h"
10:
11: long lMyPort;
12:
13: void FindTMServices();
14: void DeleteTasks();
15: void CopyTasks();
16: void RegInfoHandler(const SDMRegInfo&);
17:
18: const int ID_START_TASK = 1;
19:
20: SDMMMessage_ID idStartTask;
21: SDMComponent_ID idTaskManager;
22:
23: bool bFinish = false;
24:
25: int main (int argc, char** argv)
26: {
27:     SDMInit(argc, argv);
28:
29:     lMyPort = getPort();
30:     if (lMyPort == SDM_PM_NOT_AVAILABLE)
31:         lMyPort = 5432;
32:
33:     MessageManager mm;
34:     mm.Async_Init(lMyPort);
35:
36:     CopyTasks();
37:     FindTMServices();
38:
39:     char buf[BUFSIZE];
```

```

40: SDMRegInfo msgRegInfo;
41: while (!bFinish)
42: {
43:     if (mm.IsReady())
44:     {
45:         switch(mm.GetMessage(buf))
46:         {
47:             case SDM_RegInfo:
48:             {
49:                 if (msgRegInfo.Unmarshal(buf) != SDM_NO_FURTHER_DATA_PROVIDER)
50:                 {
51:                     RegInfoHandler(msgRegInfo);
52:                 }
53:                 break;
54:             }
55:         }
56:     }
57:     else
58:         sleep(1);
59: }
60:
61: return 0;
62: }
63:
64: void RegInfoHandler(const SDMRegInfo& msgInfo)
65: {
66: if (msgInfo.id == ID_START_TASK)
67: {
68:     idStartTask = msgInfo.msg_id;
69:     idTaskManager = msgInfo.source;
70:
71:     SDMCommand msgCommand;
72:     msgCommand.command_id = idStartTask;
73:     msgCommand.source = idTaskManager;
74:     strcpy(msgCommand.data + 2, "AppFailSilentTestConsumer");
75:     msgCommand.length = 28;
76:
77:     printf("Starting AppFailSilentTestConsumer \n");
78:     msgCommand.Send();
79:
80:     strcpy(msgCommand.data + 2, "AppFailSilentTestProducer");

```

```

81:
82:     printf("Starting AppFailSilentTestProducer \n");
83:     msgCommand.Send();
84:
85:     bFinish = true;
86: }
87:
88:
89: }
90:
91: void FindTMServices()
92: {
93:     SDMReqReg msgRequest;
94:
95:     strcpy(msgRequest.device, "TaskManager");
96:     strcpy(msgRequest.item_name, "StartTask");
97:     msgRequest.id = ID_START_TASK;
98:     msgRequest.destination.setPort(lMyPort);
99:     msgRequest.Send();
100: }
101:
102: void CopyTasks()
103: {
104:     system("cp AppFailSilentTestConsumer ../../tm/");
105:     system("cp AppFailSilentTestProducer ../../tm/");
106: }
107:
108: void DeleteTasks()
109: {
110:     system("rm -f ../../tm/AppFailSilentTestConsumer");
111:     system("rm -f ../../tm/AppFailSilentTestProducer");
112: }
113:

```

File: sdm/app/test/PMTests/AppFailSilentTestProducer.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMSubreqst.h"
3: #include "../common/message/SDMDeletesub.h"
4: #include "../common/message/SDMCancelxTEDS.h"
5: #include "../common/SubscriptionManager/SubscriptionManager.h"
6: #include "../common/MessageManager/MessageManager.h"
7: #include "../common/message/SDMHeartbeat.h"
8: #include <string.h>
9: #include <sys/types.h>
10: #include <sys/stat.h>
11: #include <fcntl.h>
12: #include <unistd.h>
13: #include <stdio.h>
14: #include <stdlib.h>
15: #include <pthread.h>
16:
17: const char* XML_HEADER = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n";
18: const char* XTEDS_HEADER = "<xTEDS version= \"2.0 \" name= \"Producer_xTEDS \">";
19: const char* APP_SECTION = " \n \t<Application name= \"producer \" kind= \"data \"/>";
20: const char* INTERFACE_SECTION = " \n \t<Interface name= \"Producer_Interface \" id= \"1 \">";
21: const char* VAR_DATA_1 = " \n \t<Variable name= \"data \" format= \"UINT16 \" ";
22: const char* VAR_DATA_2 = "kind= \"data \"/>";
23: const char* NOTIFICATION = " \n \t<Notification>";
24: const char* MSG_ALL_1 = " \n \t \t<DataMsg name= \"all \" id= \"1 \" ";
25: const char* MSG_ALL_2 = "msgArrival= \"PERIODIC \" msgRate= \"1 \">";
26: const char* MSG_ALL_3 = " \n \t \t \t<VariableRef name= \"data \"/>";
27: const char* MSG_ALL_4 = " \n \t \t</DataMsg>";
28: const char* NOTIFICATION_END = " \n \t</Notification>";
29: const char* INTERFACE_END = " \n \t</Interface>";
30: const char* XTEDS_END = " \n</xTEDS> \n";
31:
32: void RegisterxTEDS();
33: void CancelxTEDS();
34: void* Publisher(void *);
35: void* Listener(void *);
36:
37: SubscriptionManager subscriptions;
38: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER;
39: long my_port;
```

```

40: const unsigned int THREAD_STACK_SIZE = 128000;
41:
42: int main(int argc, char** argv)
43: {
44:     pthread_t ListenerThread;
45:     pthread_t PublisherThread;
46:
47:     SDMInit(argc, argv);
48:     my_port = getPort();
49:     if(my_port == SDM_PM_NOT_AVAILABLE)
50:     {
51:         printf("No PM is available to get port from! \n");
52:         return 0;
53:     }
54:
55:     pthread_attr_t threadAttr;
56:     pthread_attr_init(&threadAttr);
57:     pthread_attr_setstacksize(&threadAttr, THREAD_STACK_SIZE);
58:
59:     pthread_create(&ListenerThread, &threadAttr, &Listener, NULL);
60:     //usleep(100);
61:     RegisterxTEDS();
62:     pthread_create(&PublisherThread, &threadAttr, &Publisher, NULL);
63:
64:     pthread_join(PublisherThread, NULL);
65:     pthread_join(ListenerThread, NULL);
66: }
67:
68: void* Publisher(void * args)
69: {
70:     int published = 0;
71:     short data;
72:     while(1)
73:     {
74:         data = (short)(rand() & 0x00FF);
75:         pthread_mutex_lock(&subscription_mutex);
76:         if (subscriptions.Publish(1, 1, (char*)&data, 2))
77:         {
78:             published++;
79:         }
80:         pthread_mutex_unlock(&subscription_mutex);

```

```

81:     printf("Produced %d \tPublished %d \n",data,published);
82:     sleep(1);
83: }
84: return NULL;
85: }
86:
87: void* Listener(void * args)
88: {
89:     char buf[BUFSIZE];
90:     SDMSubreqst sub;
91:     SDMDeletesub del;
92:     MessageManager mm;
93:     mm.Async_Init(my_port);
94:
95:     // Send one heartbeat, let the app fail
96:     SendHeartbeat();
97:
98:     while(1)
99:     {
100:         if(mm.IsReady())
101:         {
102:             //SendHeartbeat();
103: #ifdef WIN32
104:             switch(mm.GetMsg(buf))
105: #else
106:             switch(mm.GetMessage(buf))
107: #endif
108:             {
109:                 case SDM_Subreqst:
110:                     sub.Unmarshal(buf);
111:                     printf("Subscription Rec'd for %d \n",sub.msg_id.getInterfaceMessagePair());
112:                     fflush(NULL);
113:                     pthread_mutex_lock(&subscription_mutex);
114:                     subscriptions.AddSubscription(sub);
115:                     pthread_mutex_unlock(&subscription_mutex);
116:                     break;
117:                 case SDM_Deletesub:
118:                     printf("Cancel Rec'd \n");
119:                     del.Unmarshal(buf);
120:                     pthread_mutex_lock(&subscription_mutex);
121:                     subscriptions.RemoveSubscription(del);

```

```

121:         pthread_mutex_unlock(&subscription_mutex);
122:         break;
123:     default:
124:         printf("Invalid Message found! \n");
125:         fflush(NULL);
126:         break;
127:     }
128: }
129: else
130: {
131:     usleep(100000);
132: }
133: }
134: return NULL;
135: }
136:
137: void RegisterxTEDS()
138: {
139:     // create an xTEDS registration message
140:     SDMxTEDS xteds;
141:
142:     // set xTEDS
143:     strcat (xteds.xTEDS,XML_HEADER);
144:     strcat (xteds.xTEDS,XTEDS_HEADER);
145:     strcat (xteds.xTEDS,APP_SECTION);
146:     strcat (xteds.xTEDS,INTERFACE_SECTION);
147:     strcat (xteds.xTEDS,VAR_DATA_1);
148:     strcat (xteds.xTEDS,VAR_DATA_2);
149:     strcat (xteds.xTEDS,NOTIFICATION);
150:     strcat (xteds.xTEDS,MSG_ALL_1);
151:     strcat (xteds.xTEDS,MSG_ALL_2);
152:     strcat (xteds.xTEDS,MSG_ALL_3);
153:     strcat (xteds.xTEDS,MSG_ALL_4);
154:     strcat (xteds.xTEDS,NOTIFICATION_END);
155:     strcat (xteds.xTEDS,INTERFACE_END);
156:     strcat (xteds.xTEDS,XTEDS_END);
157:
158:     // set the id of this application
159:     xteds.source.setSensorID(1);
160:     xteds.source.setPort(my_port);
161:     printf("Registering producer xTEDS on port %ld \n",my_port);

```

```
162: // register with the SDM
163: xteds.Send();
164: }
165:
166: void CancelxTEDS()
167: {
168:     SDMCancelxTEDS cancel;
169:     printf("Canceling xTEDS \n");
170:     cancel.source.setSensorID(1);
171:     cancel.source.setPort(my_port);
172:     cancel.Send();
173: }
```


File: sdm/app/test/DMTests/SegmentedxTEDSTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5: #include <sys/types.h>
6: #include <sys/stat.h>
7: #include <fcntl.h>
8: #include <pthread.h>
9:
10: #include "../common/message/SDMxTEDS.h"
11: #include "../common/message/SDMCancelxTEDS.h"
12: #include "../common/message/SDMComponent_ID.h"
13: #include "../common/MessageManager/MessageManager.h"
14:
15: long Port = 4058;
16: pthread_mutex_t PortMutex = PTHREAD_MUTEX_INITIALIZER;
17: //const char *InputFilename = "LargexTEDSSmaller.xml";
18: const int MAX_READ_SIZE = 7500;
19: void *xTEDSSender(void *InputFilename);
20:
21: int main (int argc, char ** argv)
22: {
23:     if (argc < 4)
24:     {
25:         printf("Usage: %s TM-ADDR DM-ADDR PID \n",argv[0]);
26:         return 0;
27:     }
28:     SDMInit(argc, argv);
29:     pthread_t t1, t2, t3;
30:
31:     pthread_create(&t1, NULL, xTEDSSender, strdup("Battery.xml"));
32:     pthread_create(&t2, NULL, xTEDSSender, strdup("LargexTEDSSmaller.xml"));
33:     pthread_create(&t3, NULL, xTEDSSender, strdup("SolarArray.xml"));
34:
35:     pthread_join(t1, NULL);
36:     pthread_join(t2, NULL);
37:     pthread_join(t3, NULL);
38:
39:     return 0;
```

```

40: }
41:
42: void *xTEDSSender(void *Arg)
43: {
44: char *InputFilename = (char*)Arg;
45: MessageManager mm;
46: pthread_mutex_lock(&PortMutex);
47: long myPort = Port++;
48: pthread_mutex_unlock(&PortMutex);
49:
50: mm.Async_Init(myPort);
51: int FileFD;
52: printf("Reading %s...",InputFilename);
53: if ((FileFD = open(InputFilename, O_RDONLY)) < 0)
54: {
55:     perror("Error opening input file \n");
56:     return 0;
57: }
58: //
59: // Read in the entire file
60: char FileBuf[65535];
61: int FileLength = read(FileFD, FileBuf, sizeof(FileBuf));
62: if (FileLength < 0)
63: {
64:     perror("Error reading file. \n");
65:     return 0;
66: }
67: printf("File Read. \n");
68: //
69: // Figure out how many segments to send
70: int NumSegments = static_cast<int>((FileLength % MAX_READ_SIZE) == 0 ?
FileLength/static_cast<double>(MAX_READ_SIZE)
FileLength/static_cast<double>(MAX_READ_SIZE)+1);
71: SDMXTEDS xTEDSMessage;
72: xTEDSMessage.source.setPort(myPort);
73: xTEDSMessage.source.setSensorID(1);
74: for (int i = 0; i < NumSegments; i++)
75: {
76:     //Set the segment numbers in the xTEDS message
77:     xTEDSMessage.xTEDS[0] = static_cast<unsigned char>(i);
78:     xTEDSMessage.xTEDS[1] = static_cast<unsigned char>(NumSegments);

```

```

79:
80:     //Read in a portion of the xTEDS message
81:     memset(xTEDSMessage.xTEDS+2, 0, sizeof(xTEDSMessage.xTEDS)-2);
82:     strncpy(xTEDSMessage.xTEDS+2, FileBuf + i*MAX_READ_SIZE, MAX_READ_SIZE);
83:     printf("%ld - Sending segment %hhd of %hhd \n",myPort, xTEDSMessage.xTEDS[0],
xTEDSMessage.xTEDS[1]);
84:     xTEDSMessage.Send();
85:     printf("%ld - Sent. \n",myPort);
86:     //
87:     //Wait for an ACK
88:     bool ACKRecd = false;
89:     char MessageBuf[BUFSIZE];
90:     while (!ACKRecd)
91:     {
92:         usleep(1000);
93:         if (mm.GetMessage(MessageBuf) == SDM_ACK)
94:         {
95:             printf("%ld - ACK \n", myPort);
96:             ACKRecd = true;
97:         }
98:         else
99:             printf(".");
100:         fflush(NULL);
101:     }
102: }
103:
104: printf("%ld - xTEDS sent. \n",myPort);
105: free (Arg);
106: usleep(200000);
107: //Send a cancel message
108: SDMCancelxTEDS cancel;
109: cancel.source.setPort(myPort);
110: cancel.source.setSensorID(1);
111: cancel.Send();
112:
113: close(FileFD);
114:
115: return NULL;
116: }

```

File: sdm/app/test/DMTTests/SolarArray.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                      xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="SolarArray_XTEDS"
4:   version="2.0">
5:   <Device name="SolarArray" kind="sa" description="a device that provides electrical power" >
6:     <Qualifier name="Manufacturer" value="MsiFitsArray"/>
7:     <Qualifier name="Model" value="1.2.3"/>
8:     <Qualifier name="SerialNumber" value="90210"/>
9:   </Device>
10:
11:   <Interface name="SolarArrayInterface" id="1">
12:     <Qualifier name="CellType" value="GaAs"/>
13:     <Qualifier name="BOLPower" value="80" units="W"/>
14:     <Qualifier name="PerformanceDegradation" value="5" units="percent_year"/>
15:     <Qualifier name="MaxPowerVoltage" value="20" units="Volts"/>
16:     <Qualifier name="MaxPowerCurrent" value="10" units="Amps"/>
17:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
18:     <Variable   kind="SubSeconds"   name="SubS"   units="Counts"   format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
19:     <Variable   name="SATemp"   kind="Temperature"   format="FLOAT32"   units="degC"
description="ambient temperature of array" />
20:     <Variable   name="SACurrent"   kind="Current"   format="FLOAT32"   units="Amps"
description="charging is positive; discharging is negative" />
21:     <Variable   name="SADesiredCurrent"   kind="SetCurrent"   format="FLOAT32"   units="Amps"
description="charging is positive; discharging is negative" />
22:     <Variable   name="SAUnregVoltage"   kind="Voltage"   format="FLOAT32"   units="Volts"
description="voltage at battery terminals" />
23:     <Variable name="SAOperationState" kind="mode" format="UINT08">
24:       <Drange name="OpStateEnum">
25:         <Option name="Offline" value="0"/>
26:         <Option name="Online" value="1"/>
27:       </Drange>
28:     </Variable>
29:
30:   <Command>
31:     <CommandMsg id="1" name="SetOpState">
32:       <VariableRef name="SAOperationState"/>
33:     </CommandMsg>
```

```

34:     </Command>
35:
36:     <Command>
37:         <CommandMsg id="2" name="SetCurrentLimitOut">
38:             <VariableRef name="SADesiredCurrent"/>
39:         </CommandMsg>
40:     </Command>
41:
42:     <Notification>
43:         <DataMsg      name="ArraySOH"      description="state    of    health"    id="3"
msgArrival="PERIODIC">
44:             <Qualifier name="telemetryLevel" value="1"/>
45:             <Qualifier name="Type" value="SOH" />
46:             <VariableRef name="Time" />
47:             <VariableRef name="SubS" />
48:             <VariableRef name="SATemp" />
49:             <VariableRef name="SACurrent" />
50:             <VariableRef name="SADesiredCurrent"/>
51:             <VariableRef name="SAUnregVoltage" />
52:             <VariableRef name="SAOperationState" />
53:         </DataMsg>
54:     </Notification>
55:
56: </Interface>
57:
58: <Interface id="2" name="DeployerInterface" description="Interface for a deployment mechanism - in
this case the release for the array.">
59:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
60:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
61:     <Variable kind="mode" name="ArmedState" format="UINT08">
62:         <Drange name="ArmedStateEnum">
63:             <Option name="Safe" value="0"/>
64:             <Option name="Armed" value="1"/>
65:         </Drange>
66:     </Variable>
67:     <Variable kind="mode" name="DeployedState" format="UINT08">
68:         <Drange name="DeployedStateEnum">
69:             <Option name="Stowed" value="0"/>
70:             <Option name="Deploying" value="1"/>
71:             <Option name="Deployed" value="2"/>
72:         </Drange>

```

```

73: </Variable>
74: <Command>
75:   <CommandMsg id="1" name="ArmOrDisarmRelease" description="Toggles the safe/armed state
of the deployment mechanism">
76:     <VariableRef name="ArmedState"/>
77:   </CommandMsg>
78: </Command>
79: <Command>
80:   <CommandMsg id="2" name="DeployGo" description="Deploy the array"/>
81: </Command>
82: <Request>
83:   <CommandMsg id="3" name="GetDeployState" />
84:   <DataReplyMsg id="4" name="DeployState">
85:     <VariableRef name="Time"/>
86:     <VariableRef name="SubS"/>
87:     <VariableRef name="ArmedState"/>
88:     <VariableRef name="DeployedState"/>
89:   </DataReplyMsg>
90: </Request>
91: </Interface>
92:
93: <Interface name="CmpSafety" id="3">
94:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
95:   <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
96:   <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
97:   <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
98:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
99:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
100:   <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32"
units="degC" />
101:   <Request>
102:     <CommandMsg name="GetDeviceTemperature" id="1" />
103:     <DataReplyMsg name="DeviceTempReply" id="2">
104:       <VariableRef name="Time" />
105:       <VariableRef name="SubS" />
106:       <VariableRef name="DeviceTemperature"/>
107:     </DataReplyMsg>
108:   </Request>
109:   <Notification>
110:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
111:       <Qualifier name="telemetryLevel" value="1"/>

```

```

112:         <VariableRef name="Time" />
113:         <VariableRef name="SubS" />
114:         <VariableRef name="DeviceTemperature"/>
115:     </DataMsg>
116: </Notification>
117: </Interface>
118: <Interface name="SolarArrayInterface" id="4">
119:     <Qualifier name="CellType" value="GaAs"/>
120:     <Qualifier name="BOLPower" value="80" units="W"/>
121:     <Qualifier name="PerformanceDegradation" value="5" units="percent_year"/>
122:     <Qualifier name="MaxPowerVoltage" value="20" units="Volts"/>
123:     <Qualifier name="MaxPowerCurrent" value="10" units="Amps"/>
124:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
125:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
126:     <Variable name="SATemp" kind="Temperature" format="FLOAT32" units="degC"
description="ambient temperature of array" />
127:     <Variable name="SACurrent" kind="Current" format="FLOAT32" units="Amps"
description="charging is positive; discharging is negative" />
128:     <Variable name="SADesiredCurrent" kind="SetCurrent" format="FLOAT32" units="Amps"
description="charging is positive; discharging is negative" />
129:     <Variable name="SAUnregVoltage" kind="Voltage" format="FLOAT32" units="Volts"
description="voltage at battery terminals" />
130:     <Variable name="SAOperationState" kind="mode" format="UINT08">
131:         <Drange name="OpStateEnum">
132:             <Option name="Offline" value="0"/>
133:             <Option name="Online" value="1"/>
134:         </Drange>
135:     </Variable>
136:
137:     <Command>
138:         <CommandMsg id="1" name="SetOpState">
139:             <VariableRef name="SAOperationState"/>
140:         </CommandMsg>
141:     </Command>
142:
143:     <Command>
144:         <CommandMsg id="2" name="SetCurrentLimitOut">
145:             <VariableRef name="SADesiredCurrent"/>
146:         </CommandMsg>
147:     </Command>
148:

```

```

149:      <Notification>
150:          <DataMsg      name="ArraySOH"      description="state      of      health"      id="3"
msgArrival="PERIODIC">
151:              <Qualifier name="telemetryLevel" value="1"/>
152:              <Qualifier name="Type" value="SOH" />
153:              <VariableRef name="Time" />
154:              <VariableRef name="SubS" />
155:              <VariableRef name="SATemp" />
156:              <VariableRef name="SACurrent" />
157:              <VariableRef name="SADesiredCurrent"/>
158:              <VariableRef name="SAUnregVoltage" />
159:              <VariableRef name="SAOperationState" />
160:          </DataMsg>
161:      </Notification>
162:
163:  </Interface>
164:  <Interface name="SolarArrayInterface" id="5">
165:      <Qualifier name="CellType" value="GaAs"/>
166:      <Qualifier name="BOLPower" value="80" units="W"/>
167:      <Qualifier name="PerformanceDegradation" value="5" units="percent_year"/>
168:      <Qualifier name="MaxPowerVoltage" value="20" units="Volts"/>
169:      <Qualifier name="MaxPowerCurrent" value="10" units="Amps"/>
170:      <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
171:      <Variable      kind="SubSeconds"      name="SubS"      units="Counts"      format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
172:      <Variable      name="SATemp"      kind="Temperature"      format="FLOAT32"      units="degC"
description="ambient temperature of array" />
173:      <Variable      name="SACurrent"      kind="Current"      format="FLOAT32"      units="Amps"
description="charging is positive; discharging is negative" />
174:      <Variable name="SADesiredCurrent" kind="SetCurrent" format="FLOAT32" units="Amps"
description="charging is positive; discharging is negative" />
175:      <Variable      name="SAUnregVoltage"      kind="Voltage"      format="FLOAT32"      units="Volts"
description="voltage at battery terminals" />
176:      <Variable name="SAOperationState" kind="mode" format="UINT08">
177:          <Drange name="OpStateEnum">
178:              <Option name="Offline" value="0"/>
179:              <Option name="Online" value="1"/>
180:          </Drange>
181:      </Variable>
182:
183:      <Command>
184:          <CommandMsg id="1" name="SetOpState">

```



```

185:         <VariableRef name="SAOperationState"/>
186:     </CommandMsg>
187: </Command>
188:
189: <Command>
190:     <CommandMsg id="2" name="SetCurrentLimitOut">
191:         <VariableRef name="SADesiredCurrent"/>
192:     </CommandMsg>
193: </Command>
194:
195: <Notification>
196:     <DataMsg      name="ArraySOH"      description="state    of    health"    id="3"
msgArrival="PERIODIC">
197:         <Qualifier name="telemetryLevel" value="1"/>
198:         <Qualifier name="Type" value="SOH" />
199:         <VariableRef name="Time" />
200:         <VariableRef name="SubS" />
201:         <VariableRef name="SATemp" />
202:         <VariableRef name="SACurrent" />
203:         <VariableRef name="SADesiredCurrent"/>
204:         <VariableRef name="SAUnregVoltage" />
205:         <VariableRef name="SAOperationState" />
206:     </DataMsg>
207: </Notification>
208:
209: </Interface>
210: </xTEDS>
211:

```

File: sdm/app/test/DMTTests/GenericVarReqTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <signal.h>
4: #include <unistd.h>
5: #include "../common/message/SDMVarReq.h"
6: #include "../common/message/SDMVarInfo.h"
7: #include "../common/MessageManager/MessageManager.h"
8:
9:
10: void SigHandler(int signum);
11: long myPort;
12:
13: SDMVarReq Request;
14:
15: bool Menu();
16:
17: int main (int argc, char ** argv)
18: {
19:     MessageManager mm;
20:     char buf[BUFSIZE];
21:     SDMVarInfo varinfo_msg;
22:     unsigned char type;
23:     SDMInit(argc, argv);
24:     signal(SIGINT, SigHandler);
25:     char InfoSource[40];
26:
27:     myPort = getPort();
28:     if (myPort == SDM_PM_NOT_AVAILABLE)
29:     {
30:         myPort = 4059;
31:     }
32:     mm.Async_Init(myPort);
33:
34:     printf("-----SDMVarReq Tester----- \n");
35:
36:     bool MenuFinished = false;
37:
38:     while (!MenuFinished)
39:         MenuFinished = Menu();
```

```

40:
41: printf("Sending request... \n");
42: Request.destination.setPort(myPort);
43: Request.Send();
44:
45: int count = 1;
46: bool Finished = false;
47: if (Request.reply > SDM_VARREQ_CURRENT)
48: {
49:     printf("Press CTRL+C to quit. \n");
50:     sleep(1);
51: }
52: while (!Finished || Request.reply > SDM_VARREQ_CURRENT)
53: {
54:     type = mm.BlockGetMessage(buf);
55:     switch (type)
56:     {
57:         case SDM_VarInfo:
58:         {
59:             long length = varinfo_msg.Unmarshal(buf);
60:             if (length == SDM_NO_FURTHER_DATA_PROVIDER)
61:                 Finished = true;
62:             else
63:             {
64:                 varinfo_msg.source.IDToString(InfoSource, sizeof(InfoSource));
65:                 printf("(%d) VarInfo received (%s): \n",count++, InfoSource);
66:                 printf("\t---xTEDS Portion--- \n");
67:                 printf("%s \n \n",varinfo_msg.var_xTEDS);
68:             }
69:             break;
70:         }
71:         default:
72:             printf("Unexpected message (%d). \n",type);
73:     }
74: }
75: printf("End of SDMVarInfo messages. \n");
76: printf("-----SDMVarReq Tester Finished----- \n");
77: return 0;
78: }
79:
80: bool Menu()

```

```

81: {
82: unsigned char Item;
83: printf("(1) Change \"reply \" \"\n");
84: printf("(2) Change \"msg_id \" \"\n");
85: printf("(3) Change \"id \" \"\n");
86: printf("(4) Change \"variable \" \"\n");
87: printf("(5) Submit SDMVarReq\n");
88: //Get the selection
89: printf("->");
90: scanf("%hhu",& Item);
91:
92: switch(Item)
93: {
94:     case 1:      //Input for reply
95:     {
96:         int ReplyType;
97:         printf(" \nReply Type: \n");
98:         printf("(1) \tSDM_VARREQ_CURRENT\n");
99:         printf("(2) \tSDM_VARREQ_CURRENT_AND_FUTURE\n");
100:         printf("(3) \tSDM_VARREQ_CANCEL\n");
101:         printf("->");
102:         scanf("%d",&ReplyType);
103:
104:         switch (ReplyType)
105:         {
106:             case 1:
107:                 Request.reply = SDM_VARREQ_CURRENT;
108:                 break;
109:             case 2:
110:                 Request.reply = SDM_VARREQ_CURRENT_AND_FUTURE;
111:                 break;
112:             case 3:
113:                 Request.reply = SDM_VARREQ_CANCEL;
114:                 break;
115:             default:
116:                 printf("Invalid choice, using SDM_VARREQ_CURRENT. \n");
117:                 break;
118:         }
119:     }
120:     break;
121:     case 2:      //Input for msg_id

```

```

122:     {
123:         SDMMMessage_ID MessageID;
124:         unsigned char ID;
125:         printf(" \nEnter the \"msg_id\" interface id: ");
126:         scanf("%hhu",&ID);
127:         MessageID.setInterface(ID);
128:
129:         Request.msg_id = MessageID;
130:     }
131:     break;
132:     case 3:        //Input for device
133:     {
134:         printf(" \nEnter the \"id \": ");
135:         scanf("%hd", &Request.id);
136:     }
137:     break;
138:     case 4:        //Input for interface
139:     {
140:         printf(" \nEnter the \"variable \": ");
141:         scanf("%s", Request.variable);
142:     }
143:     break;
144:     case 5:        //Finish the message building
145:     {
146:         return true;
147:     }
148:     break;
149:     default:
150:     break;
151: }
152: //User did not finish
153: return false;
154: }
155:
156: void SigHandler(int signum)
157: {
158:
159:     _exit(0);
160: }

```

File: sdm/app/test/DMTests/LargexTEDSTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5: #include <sys/types.h>
6: #include <sys/stat.h>
7: #include <fcntl.h>
8: #include <pthread.h>
9:
10: #include "../common/message/SDMxTEDS.h"
11: #include "../common/message/SDMCancelxTEDS.h"
12: #include "../common/message/SDMComponent_ID.h"
13: #include "../common/MessageManager/MessageManager.h"
14: #include "xTEDSPoster.h"
15:
16: const char* strFilename = "LargexTEDSSmallest.xml";
17:
18: int main (int argc, char ** argv)
19: {
20:     if (argc < 4)
21:     {
22:         printf("Usage: %s TM-ADDR DM-ADDR PID \n",argv[0]);
23:         return 0;
24:     }
25:     SDMInit(argc, argv);
26:
27:     printf("BUFSIZE=%d \n", BUFSIZE);
28:     printf("LARGE_MSG_BUFSIZE=%d \n", LARGE_MSG_BUFSIZE);
29:
30:     xTEDSPoster::PostxTEDS( strFilename );
31:
32:     printf("xTEDS message sent. \n");
33:     return 0;
34: }
```

File: sdm/app/test/DMTTests/xTEDSPoster.cpp

```
1: #include <string.h>
2: #include <unistd.h>
3: #include <sys/types.h>
4: #include <sys/stat.h>
5: #include <fcntl.h>
6: #include <stdio.h>
7:
8: #include "xTEDSPoster.h"
9: #include "../common/message/SDMxTEDS.h"
10: #include "../common/message/SDMCancelxTEDS.h"
11:
12: const long FAKE_XTEDS_PORT = 4334;
13: const char *FAKE_XTEDS_PATH = "1.2.3";
14:
15: xTEDSPoster::xTEDSPoster() {}
16: xTEDSPoster::~~xTEDSPoster() {}
17:
18: int xTEDSPoster::PostxTEDS(const char *filename)
19: {
20:     int fd = open(filename, O_RDONLY);
21:     if (fd < 0)
22:         return -1;
23:
24:     SDMxTEDS Xteds;
25:     char FileBuf[3*BUFSIZE];
26:     int result;
27:
28:     result = read(fd, FileBuf, sizeof(FileBuf));
29:     if (result < 0)
30:         return -1;
31:
32:     Xteds.source.setPort(FAKE_XTEDS_PORT);
33:     strncpy(Xteds.xTEDS, FileBuf, result);
34:     strcpy(Xteds.SPA_node, FAKE_XTEDS_PATH);
35:
36:     if ((result=Xteds.Send()) < 0)
37:     {
38:         switch (result)
39:         {
```

```

40:     case SDM_MESSAGE_SEND_ERROR:
41:         printf("xTEDSPoster::Error: Send Error \n");
42:         return 0;
43:         break;
44:     case SDM_MESSAGE_RECV_ERROR:
45:         printf("xTEDSPoster::Error: Recv Error \n");
46:         return 0;
47:         break;
48:     case SDM_INVALID_MESSAGE:
49:         printf("xTEDSPoster::Error: Invalid Message \n");
50:         return 0;
51:         break;
52:     case SDM_UNABLE_TO_REGISTER:
53:         printf("xTEDSPoster::Error: Unable to Register \n");
54:         return 0;
55:         break;
56:     case SDM_INVALID_XTEDS:
57:         printf("xTEDSPoster::Error: Invalid xTEDS \n");
58:         return 0;
59:         break;
60:     default:
61:         printf("xTEDSPoster::Error: Unknown error code %d \n",result);
62:         return 0;
63:         break;
64:     }
65: }
66: close(fd);
67: return 0;
68: }
69:
70: int xTEDSPoster::CancelxTEDS()
71: {
72:     SDMCancelxTEDS Cancel;
73:
74:     Cancel.source.setPort(FAKE_XTEDS_PORT);
75:     Cancel.Send();
76:     return 0;
77: }

```


File: sdm/app/test/DMTests/xTEDSPoster.h

```
1: #ifndef __XTEDS_POSTER_H_
2: #define __XTEDS_POSTER_H_
3:
4: #include "../common/message/SDMComponent_ID.h"
5:
6: class xTEDSPoster
7: {
8: public:
9:  xTEDSPoster();
10:  ~xTEDSPoster();
11:
12:  static int PostxTEDS(const char* filename);
13:  static int CancelxTEDS();
14: private:
15:
16: };
17:
18:
19: #endif
```

File: sdm/app/test/DMTests/Battery.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="Battery_XTEDS"
4:   version="2.0">
5:   <Device name="Battery" kind="bat" description="power storage device" >
6:     <Qualifier name="Manufacturer" value="DNetConceptBattery"/>
7:     <Qualifier name="Model" value="1.2.3"/>
8:     <Qualifier name="SerialNumber" value="90210"/>
9:   </Device>
10:
11:   <Interface name="BatteryBasicInterface" id="1">
12:     <Qualifier name="Chemistry" value="LiIon"/>
13:     <Qualifier name="Capacity" value="30.0" units="Amp-Hours"/>
14:     <Qualifier name="DepthOfDischargeLoWarn" value="20.0" units="percent"/>
15:     <Qualifier name="DepthOfDischargeLoKeepout" value="10.0" units="percent"/>
16:     <Qualifier name="CycleLimit" value="500" units="cycles"/>
17:     <Variable  name="BatSOC"   kind="dischargeFraction"  format="FLOAT32"  units="percent"
description="percent of capacity discharged" />
18:     <Variable  name="BatTemp"  kind="temperature"       format="FLOAT32"  units="degC"
description="ambient temperature inside battery" />
19:     <Variable  name="BatCycleCount" kind="count"        format="INT16"    units="counts"
description="count of discharge/recharge cycles the device has experienced" />
20:     <Variable  name="BatCurrent" kind="electricalCurrent" format="FLOAT32"  units="A"
description="charging is positive; discharging is negative" />
21:     <Variable  name="BatUnregVoltage" kind="voltage"     format="FLOAT32"  units="V"
description="voltage at battery terminals" />
22:     <Variable  name="CurrentLimitIn" kind="electricalCurrent" format="FLOAT32"  units="A"
description="Limit on current flowing into battery" />
23:     <Variable  name="CurrentLimitOut" kind="electricalCurrent" format="FLOAT32"  units="A"
description="Limit on current flowing out of battery" />
24:     <Variable name="BatteryState" kind="mode" format="UINT08" >
25:       <Drange name="BatteryStateEnum" >
26:         <Option name="Offline" value="0"/>
27:         <Option name="Online" value="1"/>
28:         <Option name="Calibrate" value="2"/>
29:       </Drange>
30:     </Variable>
31:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
```

```

32: <Variable      kind="SubSeconds"      name="SubS"      units="Counts"      format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
33:
34: <Notification>
35:     <DataMsg name="BatSOH" description="state of health" id="2" msgArrival="PERIODIC">
36:         <Qualifier name="telemetryLevel" value="1"/>
37:         <VariableRef name="Time" />
38:         <VariableRef name="SubS" />
39:         <VariableRef name="BatteryState" />
40:         <VariableRef name="BatSOC" />
41:         <VariableRef name="BatTemp" />
42:         <VariableRef name="BatCycleCount" />
43:         <VariableRef name="BatCurrent" />
44:         <VariableRef name="BatUnregVoltage" />
45:     </DataMsg>
46: </Notification>
47:
48: <Command>
49:     <CommandMsg id="2" name="SetCurrentLimitIn">
50:         <VariableRef name="CurrentLimitIn"/>
51:     </CommandMsg>
52: </Command>
53:
54: <Command>
55:     <CommandMsg id="3" name="SetCurrentLimitOut">
56:         <VariableRef name="CurrentLimitOut"/>
57:     </CommandMsg>
58: </Command>
59:
60: <Command>
61:     <CommandMsg id="4" name="SetBatteryState">
62:         <VariableRef name="BatteryState"/>
63:     </CommandMsg>
64: </Command>
65: </Interface>
66:
67: <Interface name="CmpSafety" id="3">
68:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
69:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
70:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
71:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>

```

```

72:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
73:    <Variable    kind="SubSeconds"    name="SubS"    units="Counts"    format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
74:    <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
75:    <Request>
76:        <CommandMsg name="GetDeviceTemperature" id="1" />
77:        <DataReplyMsg name="DeviceTempReply" id="2">
78:            <VariableRef name="Time" />
79:            <VariableRef name="SubS" />
80:            <VariableRef name="DeviceTemperature"/>
81:        </DataReplyMsg>
82:    </Request>
83:    <Notification>
84:        <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
85:            <Qualifier name="telemetryLevel" value="1"/>
86:            <VariableRef name="Time" />
87:            <VariableRef name="SubS" />
88:            <VariableRef name="DeviceTemperature"/>
89:        </DataMsg>
90:    </Notification>
91: </Interface>
92: <Interface name="BatteryBasicInterface" id="4">
93:     <Qualifier name="Chemistry" value="LiIon"/>
94:     <Qualifier name="Capacity" value="30.0" units="Amp-Hours"/>
95:     <Qualifier name="DepthOfDischargeLoWarn" value="20.0" units="percent"/>
96:     <Qualifier name="DepthOfDischargeLoKeepout" value="10.0" units="percent"/>
97:     <Qualifier name="CycleLimit" value="500" units="cycles"/>
98:     <Variable  name="BatSOC"   kind="dischargeFraction"  format="FLOAT32"   units="percent"
description="percent of capacity discharged" />
99:     <Variable  name="BatTemp"  kind="temperature"       format="FLOAT32"   units="degC"
description="ambient temperature inside battery" />
100:    <Variable  name="BatCycleCount"  kind="count"        format="INT16"     units="counts"
description="count of discharge/recharge cycles the device has experienced" />
101:    <Variable  name="BatCurrent"   kind="electricalCurrent" format="FLOAT32"   units="A"
description="charging is positive; discharging is negative" />
102:    <Variable  name="BatUnregVoltage" kind="voltage"      format="FLOAT32"   units="V"
description="voltage at battery terminals" />
103:    <Variable  name="CurrentLimitIn" kind="electricalCurrent" format="FLOAT32"   units="A"
description="Limit on current flowing into battery" />
104:    <Variable  name="CurrentLimitOut" kind="electricalCurrent" format="FLOAT32"   units="A"
description="Limit on current flowing out of battery" />
105:    <Variable name="BatteryState" kind="mode" format="UINT08" >
106:        <Drange name="BatteryStateEnum" >

```

```

107:         <Option name="Offline" value="0"/>
108:         <Option name="Online" value="1"/>
109:         <Option name="Calibrate" value="2"/>
110:     </Drange>
111: </Variable>
112: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
113: <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
114:
115: <Notification>
116:     <DataMsg name="BatSOH" description="state of health" id="2"
msgArrival="PERIODIC">
117:         <Qualifier name="telemetryLevel" value="1"/>
118:         <VariableRef name="Time" />
119:         <VariableRef name="SubS" />
120:         <VariableRef name="BatteryState" />
121:         <VariableRef name="BatSOC" />
122:         <VariableRef name="BatTemp" />
123:         <VariableRef name="BatCycleCount" />
124:         <VariableRef name="BatCurrent" />
125:         <VariableRef name="BatUnregVoltage" />
126:     </DataMsg>
127: </Notification>
128:
129: <Command>
130:     <CommandMsg id="2" name="SetCurrentLimitIn">
131:         <VariableRef name="CurrentLimitIn"/>
132:     </CommandMsg>
133: </Command>
134:
135: <Command>
136:     <CommandMsg id="3" name="SetCurrentLimitOut">
137:         <VariableRef name="CurrentLimitOut"/>
138:     </CommandMsg>
139: </Command>
140:
141: <Command>
142:     <CommandMsg id="4" name="SetBatteryState">
143:         <VariableRef name="BatteryState"/>
144:     </CommandMsg>
145: </Command>

```

```

146: </Interface>
147: <Interface name="BatteryBasicInterface" id="5">
148:   <Qualifier name="Chemistry" value="LiIon"/>
149:   <Qualifier name="Capacity" value="30.0" units="Amp-Hours"/>
150:   <Qualifier name="DepthOfDischargeLoWarn" value="20.0" units="percent"/>
151:   <Qualifier name="DepthOfDischargeLoKeepout" value="10.0" units="percent"/>
152:   <Qualifier name="CycleLimit" value="500" units="cycles"/>
153:   <Variable name="BatSOC" kind="dischargeFraction" format="FLOAT32" units="percent"
description="percent of capacity discharged" />
154:   <Variable name="BatTemp" kind="temperature" format="FLOAT32" units="degC"
description="ambient temperature inside battery" />
155:   <Variable name="BatCycleCount" kind="count" format="INT16" units="counts"
description="count of discharge/recharge cycles the device has experienced" />
156:   <Variable name="BatCurrent" kind="electricalCurrent" format="FLOAT32" units="A"
description="charging is positive; discharging is negative" />
157:   <Variable name="BatUnregVoltage" kind="voltage" format="FLOAT32" units="V"
description="voltage at battery terminals" />
158:   <Variable name="CurrentLimitIn" kind="electricalCurrent" format="FLOAT32" units="A"
description="Limit on current flowing into battery" />
159:   <Variable name="CurrentLimitOut" kind="electricalCurrent" format="FLOAT32" units="A"
description="Limit on current flowing out of battery" />
160:   <Variable name="BatteryState" kind="mode" format="UINT08" >
161:     <Drange name="BatteryStateEnum" >
162:       <Option name="Offline" value="0"/>
163:       <Option name="Online" value="1"/>
164:       <Option name="Calibrate" value="2"/>
165:     </Drange>
166:   </Variable>
167:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
168:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
169:
170:   <Notification>
171:     <DataMsg name="BatSOH" description="state of health" id="2"
msgArrival="PERIODIC">
172:       <Qualifier name="telemetryLevel" value="1"/>
173:       <VariableRef name="Time" />
174:       <VariableRef name="SubS" />
175:       <VariableRef name="BatteryState" />
176:       <VariableRef name="BatSOC" />
177:       <VariableRef name="BatTemp" />
178:       <VariableRef name="BatCycleCount" />
179:       <VariableRef name="BatCurrent" />

```

```

180:         <VariableRef name="BatUnregVoltage" />
181:     </DataMsg>
182: </Notification>
183:
184: <Command>
185:     <CommandMsg id="2" name="SetCurrentLimitIn">
186:         <VariableRef name="CurrentLimitIn"/>
187:     </CommandMsg>
188: </Command>
189:
190: <Command>
191:     <CommandMsg id="3" name="SetCurrentLimitOut">
192:         <VariableRef name="CurrentLimitOut"/>
193:     </CommandMsg>
194: </Command>
195:
196: <Command>
197:     <CommandMsg id="4" name="SetBatteryState">
198:         <VariableRef name="BatteryState"/>
199:     </CommandMsg>
200: </Command>
201: </Interface>
202: <Interface name="BatteryBasicInterface" id="7">
203:     <Qualifier name="Chemistry" value="LIon"/>
204:     <Qualifier name="Capacity" value="30.0" units="Amp-Hours"/>
205:     <Qualifier name="DepthOfDischargeLoWarn" value="20.0" units="percent"/>
206:     <Qualifier name="DepthOfDischargeLoKeepout" value="10.0" units="percent"/>
207:     <Qualifier name="CycleLimit" value="500" units="cycles"/>
208:     <Variable name="BatSOC" kind="dischargeFraction" format="FLOAT32" units="percent"
description="percent of capacity discharged" />
209:     <Variable name="BatTemp" kind="temperature" format="FLOAT32" units="degC"
description="ambient temperature inside battery" />
210:     <Variable name="BatCycleCount" kind="count" format="INT16" units="counts"
description="count of discharge/recharge cycles the device has experienced" />
211:     <Variable name="BatCurrent" kind="electricalCurrent" format="FLOAT32" units="A"
description="charging is positive; discharging is negative" />
212:     <Variable name="BatUnregVoltage" kind="voltage" format="FLOAT32" units="V"
description="voltage at battery terminals" />
213:     <Variable name="CurrentLimitIn" kind="electricalCurrent" format="FLOAT32" units="A"
description="Limit on current flowing into battery" />
214:     <Variable name="CurrentLimitOut" kind="electricalCurrent" format="FLOAT32" units="A"
description="Limit on current flowing out of battery" />

```

```

215:    <Variable name="BatteryState" kind="mode" format="UINT08" >
216:        <Drange name="BatteryStateEnum" >
217:            <Option name="Offline" value="0"/>
218:            <Option name="Online" value="1"/>
219:            <Option name="Calibrate" value="2"/>
220:        </Drange>
221:    </Variable>
222:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
223:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
224:
225:    <Notification>
226:        <DataMsg name="BatSOH" description="state of health" id="2"
msgArrival="PERIODIC">
227:            <Qualifier name="telemetryLevel" value="1"/>
228:            <VariableRef name="Time" />
229:            <VariableRef name="SubS" />
230:            <VariableRef name="BatteryState" />
231:            <VariableRef name="BatSOC" />
232:            <VariableRef name="BatTemp" />
233:            <VariableRef name="BatCycleCount" />
234:            <VariableRef name="BatCurrent" />
235:            <VariableRef name="BatUnregVoltage" />
236:        </DataMsg>
237:    </Notification>
238:
239:    <Command>
240:        <CommandMsg id="2" name="SetCurrentLimitIn">
241:            <VariableRef name="CurrentLimitIn"/>
242:        </CommandMsg>
243:    </Command>
244:
245:    <Command>
246:        <CommandMsg id="3" name="SetCurrentLimitOut">
247:            <VariableRef name="CurrentLimitOut"/>
248:        </CommandMsg>
249:    </Command>
250:
251:    <Command>
252:        <CommandMsg id="4" name="SetBatteryState">
253:            <VariableRef name="BatteryState"/>

```



```

254:     </CommandMsg>
255:   </Command>
256: </Interface>
257: <Interface name="BatteryBasicInterface" id="9">
258:   <Qualifier name="Chemistry" value="LiIon"/>
259:   <Qualifier name="Capacity" value="30.0" units="Amp-Hours"/>
260:   <Qualifier name="DepthOfDischargeLoWarn" value="20.0" units="percent"/>
261:   <Qualifier name="DepthOfDischargeLoKeepout" value="10.0" units="percent"/>
262:   <Qualifier name="CycleLimit" value="500" units="cycles"/>
263:   <Variable name="BatSOC" kind="dischargeFraction" format="FLOAT32" units="percent"
description="percent of capacity discharged" />
264:   <Variable name="BatTemp" kind="temperature" format="FLOAT32" units="degC"
description="ambient temperature inside battery" />
265:   <Variable name="BatCycleCount" kind="count" format="INT16" units="counts"
description="count of discharge/recharge cycles the device has experienced" />
266:   <Variable name="BatCurrent" kind="electricalCurrent" format="FLOAT32" units="A"
description="charging is positive; discharging is negative" />
267:   <Variable name="BatUnregVoltage" kind="voltage" format="FLOAT32" units="V"
description="voltage at battery terminals" />
268:   <Variable name="CurrentLimitIn" kind="electricalCurrent" format="FLOAT32" units="A"
description="Limit on current flowing into battery" />
269:   <Variable name="CurrentLimitOut" kind="electricalCurrent" format="FLOAT32" units="A"
description="Limit on current flowing out of battery" />
270:   <Variable name="BatteryState" kind="mode" format="UINT08" >
271:     <Drange name="BatteryStateEnum" >
272:       <Option name="Offline" value="0"/>
273:       <Option name="Online" value="1"/>
274:       <Option name="Calibrate" value="2"/>
275:     </Drange>
276:   </Variable>
277:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
278:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
279:
280:   <Notification>
281:     <DataMsg name="BatSOH" description="state of health" id="2"
msgArrival="PERIODIC">
282:       <Qualifier name="telemetryLevel" value="1"/>
283:       <VariableRef name="Time" />
284:       <VariableRef name="SubS" />
285:       <VariableRef name="BatteryState" />
286:       <VariableRef name="BatSOC" />
287:       <VariableRef name="BatTemp" />

```

```

288:         <VariableRef name="BatCycleCount" />
289:         <VariableRef name="BatCurrent" />
290:         <VariableRef name="BatUnregVoltage" />
291:     </DataMsg>
292: </Notification>
293:
294: <Command>
295:     <CommandMsg id="2" name="SetCurrentLimitIn">
296:         <VariableRef name="CurrentLimitIn"/>
297:     </CommandMsg>
298: </Command>
299:
300: <Command>
301:     <CommandMsg id="3" name="SetCurrentLimitOut">
302:         <VariableRef name="CurrentLimitOut"/>
303:     </CommandMsg>
304: </Command>
305:
306: <Command>
307:     <CommandMsg id="4" name="SetBatteryState">
308:         <VariableRef name="BatteryState"/>
309:     </CommandMsg>
310: </Command>
311: </Interface>
312:
313: </xTEDS>
314:

```

File: sdm/app/test/DMTests/xTEDSQualifierTests.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <signal.h>
4: #include <unistd.h>
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMRegInfo.h"
7: #include "../common/MessageManager/MessageManager.h"
8: #include "xTEDSPoster.h"
9:
10: const long myPort = 4556;
11: enum {TEST1, TEST2, TEST3, TEST4, TEST5, TEST6, TEST7, TEST8, TEST9, TEST10,
TEST11, TEST12};
12: xTEDSPoster Poster;
13: const int NUM_TESTS = 10;
14:
15: const char *Test1Correct = "<Notification> \n \t<DataMsg name= \"Test1 \" id= \"257 \" msgArrival=
\"PERIODIC \" msgRate= \"1.000000 \"> \n \t \t<Qualifier name= \"telemetryLevel1 \" value= \"1 \" /> \n
\t \t<VariableRef name= \"Time \"/> \n \t \t<VariableRef name= \"SubS \"/> \n \t \t<VariableRef name=
\"HCB_Current \"/> \n \t \t<VariableRef name= \"HCB_State \"/> \n \t</DataMsg> \n</Notification>";
16:
17: const char *Test2Correct = "<Notification> \n \t<DataMsg name= \"Test2 \" id= \"258 \" msgArrival=
\"EVENT \"> \n \t \t<Qualifier name= \"telemetryLevel2 \" value= \"2 \" /> \n \t \t<VariableRef name=
\"Time \"/> \n \t \t<VariableRef name= \"SubS \"/> \n \t \t<VariableRef name= \"HCB_State \"/> \n
\t</DataMsg> \n</Notification>";
18:
19: const char *Test3Correct = "<Command> \n \t<CommandMsg name= \"Test3 \" id= \"259 \"> \n \t
\t<Qualifier name= \"telemetryLevel3 \" value= \"3 \" /> \n \t \t<VariableRef name= \"HCB_ID \"/> \n \t
\t<VariableRef name= \"Cmd_State \"/> \n \t</CommandMsg> \n</Command>";
20:
21: const char *Test4Correct = "<Notification> \n \t<DataMsg name= \"Test4 \" id= \"513 \" msgArrival=
\"PERIODIC \" msgRate= \"1.000000 \"> \n \t \t<Qualifier name= \"telemetryLevel \" value= \"1 \" /> \n
\t \t<VariableRef name= \"Time \"/> \n \t \t<VariableRef name= \"SubS \"/> \n \t \t<VariableRef name=
\"EndpointPowerState \"/> \n \t \t<VariableRef name= \"EndpointVoltage \"/> \n \t \t<VariableRef name=
\"EndpointCurrent \"/> \n \t</DataMsg> \n</Notification>";
22:
23: const char *Test5Correct = "<Notification> \n \t<DataMsg name= \"Test5 \" id= \"514 \" msgArrival=
\"EVENT \"> \n \t \t<Qualifier name= \"telemetryLevel \" value= \"1 \" /> \n \t \t<VariableRef name=
\"Time \"/> \n \t \t<VariableRef name= \"SubS \"/> \n \t \t<VariableRef name= \"EndpointPowerState \"/>
\n \t</DataMsg> \n</Notification>";
24:
25: const char *Test6Correct = "<Command> \n \t<CommandMsg name= \"Test6 \" id= \"515 \"> \n \t
\t<Qualifier name= \"TestQual \" value= \"val \" /> \n \t \t<Qualifier name= \"TestQual2 \" value= \"val2
\" /> \n \t \t<VariableRef name= \"EndpointID \"/> \n \t \t<VariableRef name= \"SoftCurrentLimit \"/> \n
```

```
\t</CommandMsg> \n \t<FaultMsg name= \"TestFault \" id= \"546 \" description= \"This tests fault messages. \"> \n \t</FaultMsg> \n</Command>";
```

26:

```
27: const char *Test7Correct = "<Command> \n \t<CommandMsg name= \"Test7 \" id= \"516 \"> \n \t \t<Qualifier name= \"TestQual \" value= \"val \" /> \n \t \t<Qualifier name= \"TestQual2 \" value= \"val2 \" /> \n \t \t<Qualifier name= \"TestQual3 \" value= \"val3 \" /> \n \t \t<Qualifier name= \"TestQual4 \" value= \"val4 \" /> \n \t \t<VariableRef name= \"EndpointID \" /> \n \t \t<VariableRef name= \"SetCurrent \" /> \n \t</CommandMsg> \n</Command>";
```

28:

```
29: const char *Test8Correct = "<Request> \n \t<CommandMsg name= \"Test8 \" id= \"1025 \"> \n \t \t<Qualifier name= \"TestQual \" value= \"val \" /> \n \t \t<Qualifier name= \"TestQual2 \" value= \"val2 \" /> \n \t</CommandMsg> \n \t<DataMsg name= \"Test9 \" id= \"1026 \" msgArrival= \"EVENT \"> \n \t \t<Qualifier name= \"TestQual \" value= \"val \" /> \n \t \t<Qualifier name= \"TestQual2 \" value= \"val2 \" /> \n \t \t<VariableRef name= \"Time \" /> \n \t \t<VariableRef name= \"SubS \" /> \n \t \t<VariableRef name= \"TempRH \" /> \n \t \t<VariableRef name= \"TempRE \" /> \n \t</DataMsg> \n \t<FaultMsg name= \"ReqFault \" id= \"1046 \" description= \"testing request fault \"> \n \t \t<Qualifier name= \"FaultQual \" value= \"45 \" /> \n \t \t<VariableRef name= \"SubS \" /> \n \t</FaultMsg> \n</Request>";
```

30:

```
31: const char *Test9Correct = "<Request> \n \t<CommandMsg name= \"Test8 \" id= \"1025 \"> \n \t \t<Qualifier name= \"TestQual \" value= \"val \" /> \n \t \t<Qualifier name= \"TestQual2 \" value= \"val2 \" /> \n \t</CommandMsg> \n \t<DataMsg name= \"Test9 \" id= \"1026 \" msgArrival= \"EVENT \"> \n \t \t<Qualifier name= \"TestQual \" value= \"val \" /> \n \t \t<Qualifier name= \"TestQual2 \" value= \"val2 \" /> \n \t \t<VariableRef name= \"Time \" /> \n \t \t<VariableRef name= \"SubS \" /> \n \t \t<VariableRef name= \"TempRH \" /> \n \t \t<VariableRef name= \"TempRE \" /> \n \t</DataMsg> \n \t<FaultMsg name= \"ReqFault \" id= \"1046 \" description= \"testing request fault \"> \n \t \t<Qualifier name= \"FaultQual \" value= \"45 \" /> \n \t \t<VariableRef name= \"SubS \" /> \n \t</FaultMsg> \n</Request>";
```

32:

```
33: const char *Test10Correct = "<Notification> \n \t<DataMsg name= \"Test10 \" id= \"1027 \" msgArrival= \"PERIODIC \" msgRate= \"1.000000 \"> \n \t \t<Qualifier name= \"telemetryLevel \" value= \"1 \" /> \n \t \t<VariableRef name= \"Time \" /> \n \t \t<VariableRef name= \"SubS \" /> \n \t \t<VariableRef name= \"TempRH \" /> \n \t \t<VariableRef name= \"TempRE \" /> \n \t</DataMsg> \n</Notification>";
```

34:

```
35: void SigHandler(int);
```

36:

```
37: int main (int argc, char ** argv)
```

```
38: {
```

```
39:   SDMInit(argc, argv);
```

```
40:   if (Poster.PostxTEDS("xTEDSQualifierTests.xml") < 0)
```

```
41:   {
```

```
42:       printf("Error posting xTEDS. \n");
```

```
43:       return 0;
```

```
44:   }
```

```
45:   sleep(1);
```

```
46:   MessageManager mm;
```

```

47: char buf[BUFSIZE];
48: SDMRegInfo reginfo_msg;
49: SDMReqReg reqreg_msg;
50: unsigned char type;
51: SDMInit(argc, argv);
52: signal(SIGINT, SigHandler);
53: char InfoSource[40];
54:
55: mm.Async_Init(myPort);
56:
57: printf("-----Qualifiers Tester----- \n");
58:
59: printf("Sending requests... \n");
60: strcpy(reqreg_msg.device, "xTEDSQualifierTests");
61: reqreg_msg.destination.setPort(myPort);
62: strcpy(reqreg_msg.item_name, "Test1");
63: reqreg_msg.id = TEST1;
64: reqreg_msg.Send();
65: strcpy(reqreg_msg.item_name, "Test2");
66: reqreg_msg.id = TEST2;
67: reqreg_msg.Send();
68: strcpy(reqreg_msg.item_name, "Test3");
69: reqreg_msg.id = TEST3;
70: reqreg_msg.Send();
71: strcpy(reqreg_msg.item_name, "Test4");
72: reqreg_msg.id = TEST4;
73: reqreg_msg.Send();
74: strcpy(reqreg_msg.item_name, "Test5");
75: reqreg_msg.id = TEST5;
76: reqreg_msg.Send();
77: strcpy(reqreg_msg.item_name, "Test6");
78: reqreg_msg.id = TEST6;
79: reqreg_msg.Send();
80: strcpy(reqreg_msg.item_name, "Test7");
81: reqreg_msg.id = TEST7;
82: reqreg_msg.Send();
83: strcpy(reqreg_msg.item_name, "Test8");
84: reqreg_msg.id = TEST8;
85: reqreg_msg.Send();
86: strcpy(reqreg_msg.item_name, "Test9");
87: reqreg_msg.id = TEST9;

```

```

88: reqreg_msg.Send();
89: strcpy(reqreg_msg.item_name, "Test10");
90: reqreg_msg.id = TEST10;
91: reqreg_msg.Send();
92:
93: int count = 1;
94: while (count <= NUM_TESTS)
95: {
96:     type = mm.BlockGetMessage(buf);
97:     switch (type)
98:     {
99:     case SDM_RegInfo:
100:        {
101:            long length = reginfo_msg.Unmarshal(buf);
102:            if (length < 0) continue;
103:            count ++;
104:            switch(reginfo_msg.id)
105:            {
106:            case TEST1:
107:                if (strcmp(reginfo_msg.xTEDS_section, Test1Correct))
108:                {
109:                    printf("****Test1 failed.*** \n");
110:                    printf("reginfo_msg.xTEDS_section =  \"%%s \"  \n \nTest1Correct =  \"%%s \"
\n",reginfo_msg.xTEDS_section, Test1Correct);
111:                    return 0;
112:                }
113:                else
114:                    printf("Test1 passed. \n");
115:                break;
116:            case TEST2:
117:                if (strcmp(reginfo_msg.xTEDS_section, Test2Correct))
118:                {
119:                    printf("****Test2 failed.*** \n");
120:                    printf("reginfo_msg.xTEDS_section =  %%s  \n \nTest2Correct =  %%s
\n",reginfo_msg.xTEDS_section, Test2Correct);
121:                    return 0;
122:                }
123:                else
124:                    printf("Test2 passed. \n");
125:                break;
126:            case TEST3:

```

```

127:         if (strcmp(reginfo_msg.xTEDS_section, Test3Correct))
128:         {
129:             printf("****Test3 failed.*** \n");
130:             printf("reginfo_msg.xTEDS_section = %s \n \nTest3Correct = %s
\n",reginfo_msg.xTEDS_section, Test3Correct);
131:             return 0;
132:         }
133:         else
134:             printf("Test3 passed. \n");
135:         break;
136:     case TEST4:
137:         if (strcmp(reginfo_msg.xTEDS_section, Test4Correct))
138:         {
139:             printf("****Test4 failed.*** \n");
140:             printf("reginfo_msg.xTEDS_section = %s \n \nTest4Correct = %s
\n",reginfo_msg.xTEDS_section, Test4Correct);
141:             return 0;
142:         }
143:         else
144:             printf("Test4 passed. \n");
145:         break;
146:     case TEST5:
147:         if (strcmp(reginfo_msg.xTEDS_section, Test5Correct))
148:         {
149:             printf("****Test5 failed.*** \n");
150:             printf("reginfo_msg.xTEDS_section = %s \n \nTest5Correct = %s
\n",reginfo_msg.xTEDS_section, Test5Correct);
151:             return 0;
152:         }
153:         else
154:             printf("Test5 passed. \n");
155:         break;
156:     case TEST6:
157:         if (strcmp(reginfo_msg.xTEDS_section, Test6Correct))
158:         {
159:             printf("****Test6 failed.*** \n");
160:             printf("reginfo_msg.xTEDS_section = %s \n \nTest6Correct = %s
\n",reginfo_msg.xTEDS_section, Test6Correct);
161:             return 0;
162:         }
163:         else
164:             printf("Test6 passed. \n");

```

```

165:         break;
166:     case TEST7:
167:         if (strcmp(reginfo_msg.xTEDS_section, Test7Correct))
168:         {
169:             printf("****Test7 failed.*** \n");
170:             printf("reginfo_msg.xTEDS_section = %s \n \nTest7Correct = %s
\n",reginfo_msg.xTEDS_section, Test7Correct);
171:             return 0;
172:         }
173:         else
174:             printf("Test7 passed. \n");
175:         break;
176:     case TEST8:
177:         if (strcmp(reginfo_msg.xTEDS_section, Test8Correct))
178:         {
179:             printf("****Test8 failed.*** \n");
180:             printf("reginfo_msg.xTEDS_section = %s \n \nTest8Correct = %s
\n",reginfo_msg.xTEDS_section, Test8Correct);
181:             return 0;
182:         }
183:         else
184:             printf("Test8 passed. \n");
185:         break;
186:     case TEST9:
187:         if (strcmp(reginfo_msg.xTEDS_section, Test9Correct))
188:         {
189:             printf("****Test9 failed.*** \n");
190:             printf("reginfo_msg.xTEDS_section = %s \n \nTest9Correct = %s
\n",reginfo_msg.xTEDS_section, Test9Correct);
191:             return 0;
192:         }
193:         else
194:             printf("Test9 passed. \n");
195:         break;
196:     case TEST10:
197:         if (strcmp(reginfo_msg.xTEDS_section, Test10Correct))
198:         {
199:             printf("****Test10 failed.*** \n");
200:             printf("reginfo_msg.xTEDS_section = %s \n \nTest10Correct = %s
\n",reginfo_msg.xTEDS_section, Test10Correct);
201:             return 0;
202:         }

```



```

203:         else
204:             printf("Test10 passed. \n");
205:         break;
206:
207:     }
208:     reginfo_msg.source.IDToString(InfoSource, sizeof(InfoSource));
209:
210: }
211: break;
212: default:
213:     printf("Unexpected message (%d). \n",type);
214: }
215: }
216: printf("End of SDMRegInfo messages. \n");
217: printf("-----Qualifiers Tester Finished----- \n");
218: return 0;
219: }
220:
221: void SigHandler(int sig)
222: {
223:     if (sig == SIGINT)
224:     {
225:         Poster.CancelxTEDS();
226:     }
227:     _exit(0);
228: }
229:

```

File: sdm/app/test/DMTests/Makefile

```
1: include ../../Makefile.defs
2:
3:     BUILDTARGETS=SegmentedxTEDSTest      GenericReqRegTest      GenericVarReqTest
GenericSearchTest xTEDSQualifierTests xTEDSVariableTests LargexTEDSTest
4: SUBDIRS=xTEDSRegTests
5:
6: .PHONY:    all clean distclean
7:
8: all: $(BUILDTARGETS)
9:  for dir in $(SUBDIRS); do \
10:    make -C $$dir; \
11: done
12:
13: SegmentedxTEDSTest: SegmentedxTEDSTest.o
14: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
15:
16: GenericReqRegTest: GenericReqRegTest.o
17: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
18:
19: GenericVarReqTest: GenericVarReqTest.o
20: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
21:
22: GenericSearchTest: GenericSearchTest.o
23: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
24:
25: xTEDSVariableTests: xTEDSVariableTests.o xTEDSPoster.o
26: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
27:
28: xTEDSQualifierTests: xTEDSQualifierTests.o xTEDSPoster.o
29: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
30:
31: LargexTEDSTest: LargexTEDSTest.o xTEDSPoster.o
32: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
33:
34: xTEDSPoster.o: xTEDSPoster.cpp xTEDSPoster.h
35: $(CXX) $(CXXFLAGS) -c $<
36:
37: %.o: %.cpp
38: $(CXX) $(CXXFLAGS) -c $<
```

```
39:
40: clean:
41: rm -f *.o
42: for dir in $(SUBDIRS); do \
43:     make -C $$dir clean; \
44: done
45:
46: distclean: clean
47: rm -f $(BUILDTARGETS) *~
48: for dir in $(SUBDIRS); do \
49:     make -C $$dir distclean; \
50: done
```

File: sdm/app/test/DMTests/xTEDSReqRegTest.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:           xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd"
name="DNet_CoarseSSAssy_xTEDS"
4:           version="2.1">
5:
6:   <Device name="xTEDSVariableTests" kind="cssa" description="An assembly of 4 coarse sun
sensors" />
7:
8:   <Interface id="1" name="CSSAssemblyInterface" description="Aggregate-level interface for the
assembly" >
9:   <Orientation axis="X" angle="45" units="degrees"/>
10:    <Orientation axis="Y" angle="45" units="degrees"/>
11:    <Orientation axis="Z" angle="0" units="degrees"/>
12:    <Location x="5" y="5" z="-1" units="cm"/>
13:    <Variable kind="Time" name="Test1" invalidValue="123" format="UINT32" units="Seconds" />
14:    <Variable kind="SubS" name="Test2" units="Counts" format="UINT32"
scaleFactor=".00000002083" scaleUnits="Seconds" />
15:
16:    <!--Drange tests-->
17:    <Variable name="Test3" kind="boolean" format="UINT08">
18:      <Drange name="SolutionEnum" >
19:        <Option name="Yes" value="1"/>
20:      </Drange>
21:    </Variable>
22:    <Variable name="Test4" kind="boolean" invalidValue="word" format="UINT08">
23:      <Drange name="SolutionEnum" >
24:        <Option name="Test1" value="1d"/>
25:        <Option name="Test2" value="177"/>
26:        <Option name="Test3" value="156"/>
27:        <Option name="Test4" value="12"/>
28:        <Option name="Test5" value="1fe"/>
29:      </Drange>
30:    </Variable>
31:    <Notification>
32:      <DataMsg msgRate="1" msgArrival="PERIODIC" id="1" name="PowerStatus">
33:        <VariableRef name="Test1"/>
34:        <VariableRef name="Test2"/>
35:        <VariableRef name="Test3"/>
```

36: </DataMsg>
37: </Notification>
38:
39:
40: </Interface>
41: </xTEDS>

File: sdm/app/test/DMTTests/GenericReqRegTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <signal.h>
4: #include <unistd.h>
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMRegInfo.h"
7: #include "../common/MessageManager/MessageManager.h"
8:
9:
10: void SigHandler(int signum);
11: long myPort;
12:
13: SDMReqReg Request;
14:
15: bool Menu();
16:
17: int main (int argc, char ** argv)
18: {
19:     MessageManager mm;
20:     char buf[BUFSIZE];
21:     SDMRegInfo reginfo_msg;
22:     unsigned char type;
23:     char InfoSource[40];
24:     char MessageID[40];
25:     SDMInit(argc, argv);
26:     signal(SIGINT, SigHandler);
27:
28:     myPort = getPort();
29:     if (myPort == SDM_PM_NOT_AVAILABLE)
30:     {
31:         myPort = 4059;
32:     }
33:     mm.Async_Init(myPort);
34:
35:     printf("-----SDMReqReg Tester----- \n");
36:
37:     bool MenuFinished = false;
38:
39:     while (!MenuFinished)
```

```

40:     MenuFinished = Menu();
41:
42:     printf("Sending request... \n");
43:     Request.destination.setPort(myPort);
44:     Request.Send();
45:
46:     int count = 1;
47:     bool Finished = false;
48:     if (Request.reply > SDM_REQREG_CURRENT)
49:     {
50:         printf("Press CTRL+C to quit. \n");
51:         sleep(1);
52:     }
53:     while (!Finished || Request.reply > SDM_REQREG_CURRENT)
54:     {
55:         type = mm.BlockGetMessage(buf);
56:         switch (type)
57:         {
58:             case SDM_RegInfo:
59:             {
60:                 long length = reginfo_msg.Unmarshal(buf);
61:                 if (length == SDM_NO_FURTHER_DATA_PROVIDER)
62:                     Finished = true;
63:                 else
64:                 {
65:                     reginfo_msg.source.IDToString(InfoSource, sizeof(InfoSource));
66:                     reginfo_msg.msg_id.IDToString(MessageID, sizeof(MessageID));
67:                     if (reginfo_msg.type == SDM_REGINFO_REGISTRATION)
68:                         printf("(%d) RegInfo received (%s) registration (%s): \n",count++, InfoSource,
MessageID);
69:                     else if (reginfo_msg.type == SDM_REGINFO_CANCELLATION)
70:                         printf("(%d) RegInfo received (%s) cancellation (%s): \n",count++, InfoSource,
MessageID);
71:                     else
72:                         printf("(%d) RegInfo received (%s) unknown (%s): \n",count++, InfoSource,
MessageID);
73:                     printf(" \t---Message Def Portion--- \n");
74:                     printf("%s \n \n",reginfo_msg.msg_def);
75:                     printf(" \t---xTEDS Portion--- \n");
76:                     printf("%s \n \n",reginfo_msg.xTEDS_section);
77:                 }
78:                 break;

```

```

79:     }
80:     default:
81:         printf("Unexpected message (%d). \n",type);
82:     }
83: }
84: printf("End of SDMRegInfo messages. \n");
85: printf("-----SDMReqReg Tester Finished----- \n");
86: return 0;
87: }
88:
89: bool Menu()
90: {
91:     unsigned char Item;
92:     printf("(1) Change \"reply \" type \n");
93:     printf("(2) Change \"id \" \n");
94:     printf("(3) Change \"device \" \n");
95:     printf("(4) Change \"interface \" \n");
96:     printf("(5) Change \"item_name \" \n");
97:     printf("(6) Change \"quallist \" \n");
98:     printf("(7) Submit SDMReqReg \n");
99:     //Get the selection
100:     printf("->");
101:     scanf("%hhu",& Item);
102:
103:     switch(Item)
104:     {
105:         case 1:     //Input for reply
106:         {
107:             int ReplyType;
108:             printf(" \nReply Type: \n");
109:             printf("(1) \tSDM_REQREG_CURRENT \n");
110:             printf("(2) \tSDM_REQREG_CURRENT_AND_FUTURE \n");
111:             printf("(3) \tSDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS \n");
112:             printf("->");
113:             scanf("%d",&ReplyType);
114:
115:             switch (ReplyType)
116:             {
117:                 case 1:
118:                     Request.reply = SDM_REQREG_CURRENT;
119:                     break;

```



```

120:         case 2:
121:             Request.reply = SDM_REQREG_CURRENT_AND_FUTURE;
122:             break;
123:         case 3:
124:             Request.reply
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
125:             break;
126:         default:
127:             printf("Invalid choice, using SDM_REQREG_CURRENT. \n");
128:             break;
129:     }
130: }
131: break;
132: case 2:    //Input for id
133: {
134:     short Rid;
135:     printf(" \nEnter the \"id \": ");
136:     scanf("%hd",&Rid);
137:     Request.id = Rid;
138: }
139: break;
140: case 3:    //Input for device
141: {
142:     printf(" \nEnter the \"device \": ");
143:     scanf("%s", Request.device);
144: }
145: break;
146: case 4:    //Input for interface
147: {
148:     printf(" \nEnter the \"interface \": ");
149:     scanf("%s", Request.interface);
150: }
151: break;
152: case 5:    //Input for item_name
153: {
154:     printf(" \nEnter the \"item_name \": ");
155:     scanf("%s", Request.item_name);
156: }
157: break;
158: case 6:    //Input for quallist
159: {

```

```

160:         printf(" \nEnter the \"quallist \": ");
161:         scanf("%s", Request.quallist);
162:     }
163:     break;
164:     case 7:    //Finish the message building
165:     {
166:         return true;
167:     }
168:     break;
169:     default:
170:     break;
171: }
172: //User did not finish
173: return false;
174: }
175:
176: void SigHandler(int signum)
177: {
178:
179:     _exit(0);
180: }

```

File: sdm/app/test/DMTests/xTEDSVariableTests.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:           xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd"
name="DNet_CoarseSSAssy_xTEDS"
4:           version="2.1">
5:
6:   <Device name="xTEDSVariableTests" kind="cssa" description="An assembly of 4 coarse sun
sensors" />
7:
8:   <Interface id="1" name="CSSAssemblyInterface" description="Aggregate-level interface for the
assembly" >
9:   <Orientation axis="X" angle="45" units="degrees"/>
10:    <Orientation axis="Y" angle="45" units="degrees"/>
11:    <Orientation axis="Z" angle="0" units="degrees"/>
12:    <Location x="5" y="5" z="-1" units="cm"/>
13:    <Variable kind="Time" name="Test1" invalidValue="123" format="UINT32" units="Seconds" />
14:    <Variable kind="SubS" name="Test2" units="Counts" format="UINT32"
scaleFactor=".00000002083" scaleUnits="Seconds" />
15:
16:    <!--Drange tests-->
17:    <Variable name="Test3" kind="boolean" format="UINT08">
18:      <Drange name="SolutionEnum" >
19:        <Option name="Yes" value="1"/>
20:      </Drange>
21:    </Variable>
22:    <Variable name="Test4" kind="boolean" invalidValue="word" format="UINT08">
23:      <Drange name="SolutionEnum" >
24:        <Option name="Test1" value="1d"/>
25:        <Option name="Test2" value="177"/>
26:        <Option name="Test3" value="156"/>
27:        <Option name="Test4" value="12"/>
28:        <Option name="Test5" value="1fe"/>
29:      </Drange>
30:    </Variable>
31:    <Variable name="Test5" kind="boolean" format="UINT08">
32:      <Drange name="SolutionEnum" >
33:        <Option name="Yes" value="1"/>
34:        <Option name="No" value="0"/>
35:      </Drange>
```

```

36: </Variable>
37:
38:     <Variable name="Test6" kind="DataQuality" format="UINT08" rangeMin="-100.0"
rangeMax="100.0" yHigh="1.0" yLow="0.0" rHigh="20.0" rLow="10.0" precision="2"
accuracy=".0001">
39: <Qualifier name="hello" value="3" />
40: <Qualifier name="TestQual" value="45"/>
41: <Qualifier name="TestQual2" value="55" units="FakeUnits"/>
42: <Drange name="DataQualityEnum" description="None">
43: <Option name="dataBad" value="0" description="data is garbage or NaN." alarm="1"/>
44: <Option name="dataPoor" value="1"/>
45: <Option name="dataGood" value="2" description="data is good." />
46: </Drange>
47: <Location x="5" y="5" z="-1" units="cm"/>
48: <Orientation axis="X" angle="45" units="degrees"/>
49: <Orientation axis="Y" angle="45" units="degrees"/>
50: </Variable>
51: <!-- Curve tests -->
52: <Variable name="Test7" kind="SunCos" format="FLOAT32">
53: <Qualifier name="TestQual" value="45"/>
54: <Qualifier name="TestQual2" value="55" units="FakeUnits"/>
55: <Curve name="testCurve" description="Used for testing.">
56: <Coef exponent="1.0" value="1.0"/>
57: <Coef exponent="3.0" value="3.0"></Coef>
58: </Curve>
59: <Location x="5" y="5" z="-1" units="cm"/>
60: <Orientation axis="X" angle="45" units="degrees"/>
61: <Orientation axis="Y" angle="45" units="degrees"/>
62: <Orientation axis="Z" angle="0" units="degrees"/>
63: </Variable>
64: <!-- Curve tests -->
65: <Variable name="Test8" kind="SunCos" format="FLOAT32">
66: <Qualifier name="TestQual" value="45"/>
67: <Qualifier name="TestQual2" value="55" units="FakeUnits"/>
68: <Curve name="testCurve" description="Used for testing.">
69: <Coef exponent="1.0" value="14.0"/>
70: <Coef exponent="3.0" value="3.0"></Coef>
71: <Coef exponent="16.0" value="155.0"/>
72: <Coef exponent="14.0" value="143.0"/>
73: <Coef exponent="13.0" value="1662.0"/>
74: <Coef exponent="12.0" value="11.0"/>

```

```

75: </Curve>
76:   <Location x="5" y="5" z="-1" units="cm"/>
77:   <Orientation axis="Y" angle="45" units="degrees"/>
78:   <Orientation axis="Z" angle="0" units="degrees"/>
79: </Variable>
80:
81:
82:   <Variable name="Test9" kind="SunCos" format="FLOAT32">
83: <Qualifier name="TestQual" value="45"/>
84: <Qualifier name="TestQual2" value="55" units="FakeUnits"/>
85:   <Location x="5" y="0" z="-1" units="cm"/>
86:   <Orientation axis="X" angle="-45" units="degrees"/>
87:   <Orientation axis="Y" angle="45" units="degrees"/>
88:   <Orientation axis="Z" angle="0" units="degrees"/>
89: </Variable>
90: <Variable name="Test10" kind="SunCos" format="FLOAT32">
91:   <Location x="0" y="5" z="-1" units="cm"/>
92:   <Orientation axis="X" angle="45" units="degrees"/>
93:   <Orientation axis="Z" angle="0" units="degrees"/>
94: </Variable>
95: <Variable name="Test11" kind="SunCos" format="FLOAT32">
96:   <Location x="5" y="5" z="1" units="cm"/>
97:   <Orientation axis="X" angle="45" units="degrees"/>
98:   <Orientation axis="Y" angle="45" units="degrees"/>
99:   <Orientation axis="Z" angle="-90" units="degrees"/>
100: </Variable>
101: <Variable name="Test12" kind="sunLOS" format="FLOAT32" length="2" />
102:
103:
104: </Interface>
105: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSQualifierTests.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="Robust_Hub_xTEDS" version="12">
4:
5:       <Device   name="xTEDSQualifierTests"   kind="xTEDSQualifierTests"   modelId="0001"
serialNumber="12345" powerRequirements="5" >
6:   <!--<Qualifier name="Manufacturer" value=""/>
7:   <Qualifier name="Model" value="Gen1"/>-->
8: </Device>
9:
10:
11: <Interface id="1" name="RH_HCB" description="High Power Circuit Breaker">
12:   <Qualifier name="Num_HCB" value="4"/>
13:   <Qualifier name="HCB_TripCurrent" value="30.0" units="Amps"/>
14:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
15:       <Variable   kind="SubSeconds"   name="SubS"   units="Counts"   format="UINT32"
scaleFactor=".00000002083" scaleUnits="Seconds" />
16:   <Variable kind="Index" name="HCB_ID" format="UINT08" />
17:   <Variable kind="boolean" name="Cmd_State" format="UINT08" />
18:       <Variable   name="HCB_Current"   kind="Current"   length="4"   format="FLOAT32"
scaleUnits="Amps"/>
19:   <Variable name="HCB_State" kind="boolean" length="4" format="UINT08">
20:     <Drange name="HCB_StateEnum">
21:       <Option name="Off" value="0"/>
22:       <Option name="On" value="1"/>
23:       <Option name="Tripped" value="2"/>
24:     </Drange>
25:   </Variable>
26:   <Notification>
27:     <DataMsg id="1" name="Test1" msgArrival="PERIODIC" msgRate="1" >
28:       <Qualifier name="telemetryLevel1" value="1"/>
29:       <VariableRef name="Time" />
30:       <VariableRef name="SubS" />
31:       <VariableRef name="HCB_Current"/>
32:       <VariableRef name="HCB_State"/>
33:     </DataMsg>
34:   </Notification>
35:   <Notification>
```

```

36:   <DataMsg id="2" name="Test2" msgArrival="EVENT">
37:     <Qualifier name="telemetryLevel2" value="2"/>
38:     <VariableRef name="Time" />
39:     <VariableRef name="SubS" />
40:     <VariableRef name="HCB_State"/>
41:   </DataMsg>
42: </Notification>
43: <Command>
44:   <CommandMsg id="3" name="Test3">
45: <Qualifier name="telemetryLevel3" value="3"/>
46:   <VariableRef name="HCB_ID" />
47:   <VariableRef name="Cmd_State"/>
48: </CommandMsg>
49: </Command>
50: </Interface>
51:
52: <Interface id="2" name="EndpointInterface" description="Endpoint power distribution">
53:   <Qualifier name="Num_Endpoint" value="8"/>
54:   <Qualifier name="EndpointTripCurrent" value="4.5" units="Amps"/>
55:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
56:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".00000002083" scaleUnits="Seconds" />
57:   <Variable name="EndpointVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"
length="8" />
58:   <Variable name="EndpointCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"
length="8" />
59:   <Variable name="EndpointID" kind="Index" format="UINT08" />
60:   <Variable name="SetCurrent" kind="boolean" format="UINT08"/>
61:   <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32" defaultValue="1.0"
scaleUnits="Amps" />
62:   <Variable name="EndpointPowerState" kind="boolean" format="UINT08" length="8">
63:     <Drange name="PowerStateEnum">
64:       <Option name="Off" value="0"/>
65:       <Option name="On" value="1"/>
66:       <Option name="Tripped" value="2"/>
67:     </Drange>
68:   </Variable>
69: </Notification>
70: <DataMsg id="1" name="Test4" msgArrival="PERIODIC" msgRate="1" >
71:   <Qualifier name="telemetryLevel" value="1"/>
72:   <VariableRef name="Time" />
73:   <VariableRef name="SubS" />

```

```

74:     <VariableRef name="EndpointPowerState" />
75:     <VariableRef name="EndpointVoltage"/>
76:     <VariableRef name="EndpointCurrent"/>
77: </DataMsg>
78: </Notification>
79: <Notification>
80:   <DataMsg id="2" name="Test5" msgArrival="EVENT">
81:     <Qualifier name="telemetryLevel" value="1"/>
82:     <VariableRef name="Time" />
83:     <VariableRef name="SubS" />
84:     <VariableRef name="EndpointPowerState"/>
85:   </DataMsg>
86: </Notification>
87: <Command>
88:   <CommandMsg id="3" name="Test6" >
89: <Qualifier name="TestQual" value="val"/>
90: <Qualifier name="TestQual2" value="val2"/>
91:   <VariableRef name ="EndpointID" />
92:   <VariableRef name="SoftCurrentLimit"/>
93: </CommandMsg>
94:   <FaultMsg name="TestFault" description="This tests fault messages." id="34" />
95: </Command>
96: <Command>
97:   <CommandMsg id="4" name="Test7" >
98: <Qualifier name="TestQual" value="val"/>
99: <Qualifier name="TestQual2" value="val2"/>
100:   <Qualifier name="TestQual3" value="val3"/>
101:   <Qualifier name="TestQual4" value="val4"/>
102:   <VariableRef name ="EndpointID" />
103:   <VariableRef name="SetCurrent"/>
104: </CommandMsg>
105: </Command>
106: </Interface>
107:
108: <Interface name="DeviceSafety" id="4">
109:   <Variable name="TempRH" kind="temperature" format="INT16" units="degC" description="RH
Electronics" rLow="-10" yLow="0" yHigh="40" rHigh="50">
110:   <Location x="50" y="0" z="0" units="cm"/>
111: </Variable>
112:   <Variable name="TempRE" kind="temperature" format="INT16" units="degC"
description="Router Electronics" rLow="-10" yLow="0" yHigh="40" rHigh="50">

```



```

113:   <Location x="50" y="0" z="0" units="cm"/>
114: </Variable>
115:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
116:       <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".00000002083" scaleUnits="Seconds" />
117:   <Request>
118:       <CommandMsg name="Test8" id="1">
119:           <Qualifier name="TestQual" value="val"/>
120:           <Qualifier name="TestQual2" value="val2"/>
121:       </CommandMsg>
122:       <DataReplyMsg name="Test9" id="2">
123:           <Qualifier name="TestQual" value="val"/>
124:           <Qualifier name="TestQual2" value="val2"/>
125:           <VariableRef name="Time" />
126:           <VariableRef name="SubS" />
127:           <VariableRef name="TempRH"/>
128:           <VariableRef name="TempRE"/>
129:       </DataReplyMsg>
130:       <FaultMsg name="ReqFault" id="22" description="testing request fault">
131:           <Qualifier name="FaultQual" value="45"> </Qualifier>
132:           <VariableRef name="SubS"/>
133:       </FaultMsg>
134:   </Request>
135:   <Notification>
136:       <DataMsg name="Test10" id="3" msgArrival="PERIODIC" msgRate="1">
137:           <Qualifier name="telemetryLevel" value="1"/>
138:           <VariableRef name="Time" />
139:           <VariableRef name="SubS" />
140:           <VariableRef name="TempRH"/>
141:           <VariableRef name="TempRE"/>
142:       </DataMsg>
143:   </Notification>
144: </Interface>
145:
146: </xTEDS>

```

File: sdm/app/test/DMTTests/xTEDSVariableTests.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <signal.h>
4: #include <unistd.h>
5: #include "../common/message/SDMVarReq.h"
6: #include "../common/message/SDMVarInfo.h"
7: #include "../common/MessageManager/MessageManager.h"
8: #include "xTEDSPoster.h"
9:
10: const long myPort = 4756;
11: enum {TEST1, TEST2, TEST3, TEST4, TEST5, TEST6, TEST7, TEST8, TEST9, TEST10,
TEST11, TEST12};
12: xTEDSPoster Poster;
13: const int NUM_TESTS = 12;
14:
15: const char *Test1Correct = "<Variable name= \"Test1 \" kind= \"Time \" format= \"UINT32 \" units=
\"Seconds \" invalidValue= \"123 \"/>";
16:
17: const char *Test2Correct = "<Variable name= \"Test2 \" kind= \"SubS \" format= \"UINT32 \" units=
\"Counts \" scaleFactor= \".00000002083 \" scaleUnits= \"Seconds \"/>";
18:
19: const char *Test3Correct = "<Variable name= \"Test3 \" kind= \"boolean \" format= \"UINT08 \"> \n
\
20: \t<Drange name= \"SolutionEnum \"> \n \
21: \t \t<Option name= \"Yes \" value= \"1 \"/> \n \
22: \t</Drange> \n \
23: </Variable>";
24:
25: const char *Test4Correct = "<Variable name= \"Test4 \" kind= \"boolean \" format= \"UINT08 \"
invalidValue= \"word \"> \n \
26: \t<Drange name= \"SolutionEnum \"> \n \
27: \t \t<Option name= \"Test1 \" value= \"1d \"/> \n \
28: \t \t<Option name= \"Test2 \" value= \"177 \"/> \n \
29: \t \t<Option name= \"Test3 \" value= \"156 \"/> \n \
30: \t \t<Option name= \"Test4 \" value= \"12 \"/> \n \
31: \t \t<Option name= \"Test5 \" value= \"1fe \"/> \n \
32: \t</Drange> \n \
33: </Variable>";
34:
```

```

35: const char *Test5Correct = "<Variable name= \"Test5 \" kind= \"boolean \" format= \"UINT08 \"> \n
\
36: \t<Drange name= \"SolutionEnum \"> \n \
37: \t \t<Option name= \"Yes \" value= \"1 \"/> \n \
38: \t \t<Option name= \"No \" value= \"0 \"/> \n \
39: \t</Drange> \n \
40: </Variable>";
41:
42: const char *Test6Correct = "<Variable name= \"Test6 \" kind= \"DataQuality \" format= \"UINT08 \"
rangeMin= \"-100.0 \" rangeMax= \"100.0 \" precision= \"2 \" accuracy= \".0001 \" rLow= \"10.0 \"
rHigh= \"20.0 \" yLow= \"0.0 \" yHigh= \"1.0 \"> \n \
43: \t<Drange name= \"DataQualityEnum \" description= \"None \"> \n \
44: \t \t<Option name= \"dataBad \" value= \"0 \" description= \"data is garbage or NaN. \" alarm= \"1
\"/> \n \
45: \t \t<Option name= \"dataPoor \" value= \"1 \"/> \n \
46: \t \t<Option name= \"dataGood \" value= \"2 \" description= \"data is good. \"/> \n \
47: \t</Drange> \n \
48: \t<Qualifier name= \"hello \" value= \"3 \" /> \n \
49: \t \t<Qualifier name= \"TestQual \" value= \"45 \" /> \n \
50: \t \t<Qualifier name= \"TestQual2 \" value= \"55 \" units= \"FakeUnits \" /> \n \
51: \t<Location x= \"5 \" y= \"5 \" z= \"-1 \" units= \"cm \" /> \n \
52: \t<Orientation axis= \"X \" angle= \"45 \" units= \"degrees \" /> \n \
53: \t<Orientation axis= \"Y \" angle= \"45 \" units= \"degrees \" /> \n \
54: </Variable>";
55:
56: const char *Test7Correct = "<Variable name= \"Test7 \" kind= \"SunCos \" format= \"FLOAT32 \">
\n \
57: \t<Curve name= \"testCurve \" description= \"Used for testing. \"> \n \
58: \t \t<Coef exponent= \"1.0 \" value= \"1.0 \"/> \n \
59: \t \t<Coef exponent= \"3.0 \" value= \"3.0 \"/> \n \
60: \t</Curve> \n \
61: \t<Qualifier name= \"TestQual \" value= \"45 \" /> \n \
62: \t \t<Qualifier name= \"TestQual2 \" value= \"55 \" units= \"FakeUnits \" /> \n \
63: \t<Location x= \"5 \" y= \"5 \" z= \"-1 \" units= \"cm \" /> \n \
64: \t<Orientation axis= \"X \" angle= \"45 \" units= \"degrees \" /> \n \
65: \t<Orientation axis= \"Y \" angle= \"45 \" units= \"degrees \" /> \n \
66: \t<Orientation axis= \"Z \" angle= \"0 \" units= \"degrees \" /> \n \
67: </Variable>";
68:
69: const char *Test8Correct = "<Variable name= \"Test8 \" kind= \"SunCos \" format= \"FLOAT32 \">
\n \
70: \t<Curve name= \"testCurve \" description= \"Used for testing. \"> \n \

```

```

71: \t \t<Coef exponent= \"1.0 \" value= \"14.0 \"/> \n \
72: \t \t<Coef exponent= \"3.0 \" value= \"3.0 \"/> \n \
73: \t \t<Coef exponent= \"16.0 \" value= \"155.0 \"/> \n \
74: \t \t<Coef exponent= \"14.0 \" value= \"143.0 \"/> \n \
75: \t \t<Coef exponent= \"13.0 \" value= \"1662.0 \"/> \n \
76: \t \t<Coef exponent= \"12.0 \" value= \"11.0 \"/> \n \
77: \t</Curve> \n \
78: \t<Qualifier name= \"TestQual \" value= \"45 \" /> \n \
79: \t \t<Qualifier name= \"TestQual2 \" value= \"55 \" units= \"FakeUnits \" /> \n \
80: \t<Location x= \"5 \" y= \"5 \" z= \"-1 \" units= \"cm \" /> \n \
81: \t<Orientation axis= \"Y \" angle= \"45 \" units= \"degrees \" /> \n \
82: \t<Orientation axis= \"Z \" angle= \"0 \" units= \"degrees \" /> \n \
83: </Variable>";
84:
85: const char *Test9Correct = "<Variable name= \"Test9 \" kind= \"SunCos \" format= \"FLOAT32 \">
\n \
86: \t<Qualifier name= \"TestQual \" value= \"45 \" /> \n \
87: \t \t<Qualifier name= \"TestQual2 \" value= \"55 \" units= \"FakeUnits \" /> \n \
88: \t<Location x= \"5 \" y= \"0 \" z= \"-1 \" units= \"cm \" /> \n \
89: \t<Orientation axis= \"X \" angle= \"-45 \" units= \"degrees \" /> \n \
90: \t<Orientation axis= \"Y \" angle= \"45 \" units= \"degrees \" /> \n \
91: \t<Orientation axis= \"Z \" angle= \"0 \" units= \"degrees \" /> \n \
92: </Variable>";
93:
94: const char *Test10Correct = "<Variable name= \"Test10 \" kind= \"SunCos \" format= \"FLOAT32
\"> \n \
95: \t<Location x= \"0 \" y= \"5 \" z= \"-1 \" units= \"cm \" /> \n \
96: \t<Orientation axis= \"X \" angle= \"45 \" units= \"degrees \" /> \n \
97: \t<Orientation axis= \"Z \" angle= \"0 \" units= \"degrees \" /> \n \
98: </Variable>";
99:
100: const char *Test11Correct = "<Variable name= \"Test11 \" kind= \"SunCos \" format= \"FLOAT32
\"> \n \
101: \t<Location x= \"5 \" y= \"5 \" z= \"1 \" units= \"cm \" /> \n \
102: \t<Orientation axis= \"X \" angle= \"45 \" units= \"degrees \" /> \n \
103: \t<Orientation axis= \"Y \" angle= \"45 \" units= \"degrees \" /> \n \
104: \t<Orientation axis= \"Z \" angle= \"-90 \" units= \"degrees \" /> \n \
105: </Variable>";
106:
107: const char *Test12Correct = "<Variable name= \"Test12 \" kind= \"sunLOS \" format= \"FLOAT32
\" length= \"2 \"/>";
108:

```

```

109:
110: void SigHandler(int);
111:
112: int main (int argc, char ** argv)
113: {
114:     SDMInit(argc, argv);
115:     if (Poster.PostxTEDS("xTEDSVariableTests.xml") < 0)
116:     {
117:         printf("Error posting xTEDS. \n");
118:         return 0;
119:     }
120:
121:     MessageManager mm;
122:     char buf[BUFSIZE];
123:     SDMVarInfo varinfo_msg;
124:     SDMVarReq varreq_msg;
125:     unsigned char type;
126:     SDMInit(argc, argv);
127:     signal(SIGINT,SigHandler);
128:     char InfoSource[40];
129:
130:     mm.Async_Init(myPort);
131:
132:     printf("-----SDMVarReq Tester----- \n");
133:
134:     printf("Sending requests... \n");
135:
136:     varreq_msg.destination.setPort(myPort);
137:     strcpy(varreq_msg.variable, "Test1");
138:     varreq_msg.id = TEST1;
139:     varreq_msg.Send();
140:     strcpy(varreq_msg.variable, "Test2");
141:     varreq_msg.id = TEST2;
142:     varreq_msg.Send();
143:     strcpy(varreq_msg.variable, "Test3");
144:     varreq_msg.id = TEST3;
145:     varreq_msg.Send();
146:     strcpy(varreq_msg.variable, "Test4");
147:     varreq_msg.id = TEST4;
148:     varreq_msg.Send();
149:     strcpy(varreq_msg.variable, "Test5");

```

```

150:   varreq_msg.id = TEST5;
151:   varreq_msg.Send();
152:   strcpy(varreq_msg.variable, "Test6");
153:   varreq_msg.id = TEST6;
154:   varreq_msg.Send();
155:   strcpy(varreq_msg.variable, "Test7");
156:   varreq_msg.id = TEST7;
157:   varreq_msg.Send();
158:   strcpy(varreq_msg.variable, "Test8");
159:   varreq_msg.id = TEST8;
160:   varreq_msg.Send();
161:   strcpy(varreq_msg.variable, "Test9");
162:   varreq_msg.id = TEST9;
163:   varreq_msg.Send();
164:   strcpy(varreq_msg.variable, "Test10");
165:   varreq_msg.id = TEST10;
166:   varreq_msg.Send();
167:   strcpy(varreq_msg.variable, "Test11");
168:   varreq_msg.id = TEST11;
169:   varreq_msg.Send();
170:   strcpy(varreq_msg.variable, "Test12");
171:   varreq_msg.id = TEST12;
172:   varreq_msg.Send();
173:
174:   int count = 1;
175:   while (count <= NUM_TESTS)
176:   {
177:       type = mm.BlockGetMessage(buf);
178:       switch (type)
179:       {
180:           case SDM_VarInfo:
181:           {
182:               long length = varinfo_msg.Unmarshal(buf);
183:               if (length < 0) continue;
184:               count ++;
185:               switch(varinfo_msg.id)
186:               {
187:                   case TEST1:
188:                       if (strcmp(varinfo_msg.var_xTEDS, Test1Correct))
189:                           {
190:                               printf("***Test1 failed.*** \n");

```

```

191:             printf("varinfo_msg.var_xTEDS =  \\\'%s \\'  \n \nTest1Correct =  \\\'%s \\'
\n",varinfo_msg.var_xTEDS, Test1Correct);
192:             return 0;
193:         }
194:         else
195:             printf("Test1 passed. \n");
196:         break;
197:     case TEST2:
198:         if (strcmp(varinfo_msg.var_xTEDS, Test2Correct))
199:         {
200:             printf("****Test2 failed.*** \n");
201:             printf("varinfo_msg.var_xTEDS   =   %s       \n   \nTest2Correct   =   %s
\n",varinfo_msg.var_xTEDS, Test2Correct);
202:             return 0;
203:         }
204:         else
205:             printf("Test2 passed. \n");
206:         break;
207:     case TEST3:
208:         if (strcmp(varinfo_msg.var_xTEDS, Test3Correct))
209:         {
210:             printf("****Test3 failed.*** \n");
211:             printf("varinfo_msg.var_xTEDS   =   %s       \n   \nTest3Correct   =   %s
\n",varinfo_msg.var_xTEDS, Test3Correct);
212:             return 0;
213:         }
214:         else
215:             printf("Test3 passed. \n");
216:         break;
217:     case TEST4:
218:         if (strcmp(varinfo_msg.var_xTEDS, Test4Correct))
219:         {
220:             printf("****Test4 failed.*** \n");
221:             printf("varinfo_msg.var_xTEDS   =   %s       \n   \nTest4Correct   =   %s
\n",varinfo_msg.var_xTEDS, Test4Correct);
222:             return 0;
223:         }
224:         else
225:             printf("Test4 passed. \n");
226:         break;
227:     case TEST5:
228:         if (strcmp(varinfo_msg.var_xTEDS, Test5Correct))

```

```

229:        {
230:            printf("****Test5 failed.*** \n");
231:            printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest5Correct    =    %s
\n",varinfo_msg.var_xTEDS, Test5Correct);
232:            return 0;
233:        }
234:        else
235:            printf("Test5 passed. \n");
236:        break;
237:    case TEST6:
238:        if (strcmp(varinfo_msg.var_xTEDS, Test6Correct))
239:        {
240:            printf("****Test6 failed.*** \n");
241:            printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest6Correct    =    %s
\n",varinfo_msg.var_xTEDS, Test6Correct);
242:            return 0;
243:        }
244:        else
245:            printf("Test6 passed. \n");
246:        break;
247:    case TEST7:
248:        if (strcmp(varinfo_msg.var_xTEDS, Test7Correct))
249:        {
250:            printf("****Test7 failed.*** \n");
251:            printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest7Correct    =    %s
\n",varinfo_msg.var_xTEDS, Test7Correct);
252:            return 0;
253:        }
254:        else
255:            printf("Test7 passed. \n");
256:        break;
257:    case TEST8:
258:        if (strcmp(varinfo_msg.var_xTEDS, Test8Correct))
259:        {
260:            printf("****Test8 failed.*** \n");
261:            printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest8Correct    =    %s
\n",varinfo_msg.var_xTEDS, Test8Correct);
262:            return 0;
263:        }
264:        else
265:            printf("Test8 passed. \n");
266:        break;

```



```

267:         case TEST9:
268:             if (strcmp(varinfo_msg.var_xTEDS, Test9Correct))
269:             {
270:                 printf("****Test9 failed.*** \n");
271:                 printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest9Correct    =    %s\n",varinfo_msg.var_xTEDS, Test9Correct);
272:                 return 0;
273:             }
274:             else
275:                 printf("Test9 passed. \n");
276:             break;
277:         case TEST10:
278:             if (strcmp(varinfo_msg.var_xTEDS, Test10Correct))
279:             {
280:                 printf("****Test10 failed.*** \n");
281:                 printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest10Correct    =    %s\n",varinfo_msg.var_xTEDS, Test10Correct);
282:                 return 0;
283:             }
284:             else
285:                 printf("Test10 passed. \n");
286:             break;
287:         case TEST11:
288:             if (strcmp(varinfo_msg.var_xTEDS, Test11Correct))
289:             {
290:                 printf("****Test11 failed.*** \n");
291:                 printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest11Correct    =    %s\n",varinfo_msg.var_xTEDS, Test11Correct);
292:                 return 0;
293:             }
294:             else
295:                 printf("Test11 passed. \n");
296:             break;
297:         case TEST12:
298:             if (strcmp(varinfo_msg.var_xTEDS, Test12Correct))
299:             {
300:                 printf("****Test12 failed.*** \n");
301:                 printf("varinfo_msg.var_xTEDS    =    %s        \n    \nTest12Correct    =    %s\n",varinfo_msg.var_xTEDS, Test12Correct);
302:                 return 0;
303:             }
304:             else

```

```

305:             printf("Test12 passed. \n");
306:             break;
307:         }
308:         varinfo_msg.source.IDToString(InfoSource, sizeof(InfoSource));
309:         //printf("(%d) VarInfo received (%s): \n",count++, InfoSource);
310:         //printf(" \t---xTEDS Portion--- \n");
311:         //printf("%s \n \n",varinfo_msg.var_xTEDS);
312:     }
313:     break;
314:     default:
315:         printf("Unexpected message (%d). \n",type);
316:     }
317: }
318: printf("End of SDMVarInfo messages. \n");
319: printf("-----SDMVarReq Tester Finished----- \n");
320: return 0;
321: }
322:
323: void SigHandler(int sig)
324: {
325:     if (sig == SIGINT)
326:     {
327:         Poster.CancelxTEDS();
328:     }
329:     _exit(0);
330: }
331:

```

File: sdm/app/test/DMTests/LargexTEDSSmallest.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd"
name="ActivityManagerXTEDS" description="ActivityManager xTEDS" version="1.0">
4:
5: <Application name="ActivityManager" kind="AutonomyFlightSoftware" description="Autonomous
Tasking Executive (ATE), ActivityManager"/>
6:
7: <Interface name="ActivityManagerInterface" id="1">
8:
9:     <Variable name="amActivityId" kind="ID" format="UINT32"></Variable>
10:    <Variable name="amActivityType" kind="tbd" format="UINT08" length="33"></Variable><!--
33-byte, null terminated string -->
11:    <Variable name="amActivityPriority" kind="tbd" format="UINT16">
12:        <Drange name="amActivityPriorityEnum">
13:            <Option value="0" name="LOW"/>
14:            <Option value="1" name="MEDIUM"/>
15:            <Option value="2" name="HIGH"/>
16:        </Drange>
17:    </Variable>
18:    <Variable      name="amActivityBeginTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies as soon as possible -->
19:    <Variable name="amActivityDuration" kind="Time" format="FLOAT64" units="s"></Variable>
20:    <Variable      name="amActivityNotBeforeTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies now -->
21:    <Variable      name="amActivityNotAfterTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies no limit -->
22:    <Variable name="amAllowConcurrentActivity" kind="boolean" format="UINT08">
23:        <Drange name="amAllowConcurrentActivityEnum">
24:            <Option value="0" name="NO"/>
25:            <Option value="1" name="YES"/>
26:        </Drange>
27:    </Variable>
28:    <Variable name="amScheduledActivityId" kind="ID" format="UINT32"></Variable>
29:    <Variable name="amStatus" kind="Status" format="INT16">
30:        <Drange name="amStatusEnum">
31:            <Option value="0" name="FAILURE"/>
32:            <Option value="1" name="SUCCESS"/>
33:        </Drange>
```

```

34:     </Variable>
35:     <Variable name="amExecStatus" kind="Status" format="INT16">
36:         <Drange name="amExecStatusEnum">
37:             <Option value="0" name="FAILURE"/>
38:             <Option value="1" name="SUCCESS"/>
39:             <Option value="2" name="NOT_EXECUTED"/>
40:         </Drange>
41:     </Variable>
42:     <Variable name="amDebugLevel" kind="tbd" format="UINT16"></Variable>
43:     <Variable name="amCurrentDebugLevel" kind="tbd" format="UINT16"></Variable>
44:     <Variable name="amActivitiesCurrentlyScheduled" kind="tbd" format="UINT16"></Variable>
45:     <Variable name="amActivitiesExecuted" kind="tbd" format="UINT16"></Variable>
46:     <Variable name="amActivitiesExecutedSuccess" kind="tbd" format="UINT16"></Variable>
47:     <Variable name="amActivitiesExecutedFailed" kind="tbd" format="UINT16"></Variable>
48:     <Variable name="amActivitiesDeleted" kind="tbd" format="UINT16"></Variable>
49:     <Variable name="amSchedActivityReceived" kind="tbd" format="UINT16"></Variable>
50:     <Variable name="amSchedActivityAccepted" kind="tbd" format="UINT16"></Variable>
51:     <Variable name="amSchedActivitySuccess" kind="tbd" format="UINT16"></Variable>
52:     <Variable name="amSchedActivityFailure" kind="tbd" format="UINT16"></Variable>
53:     <Variable name="amAdjustActivityTimeReceived" kind="tbd" format="UINT16"></Variable>
54:     <Variable name="amAdjustActivityTimeAccepted" kind="tbd" format="UINT16"></Variable>
55:     <Variable name="amAdjustActivityTimeSuccess" kind="tbd" format="UINT16"></Variable>
56:     <Variable name="amAdjustActivityTimeFailure" kind="tbd" format="UINT16"></Variable>
57:     <Variable name="amExecuteActivityReceived" kind="tbd" format="UINT16"></Variable>
58:     <Variable name="amExecuteActivityAccepted" kind="tbd" format="UINT16"></Variable>
59:     <Variable name="amExecuteActivitySuccess" kind="tbd" format="UINT16"></Variable>
60:     <Variable name="amExecuteActivityFailure" kind="tbd" format="UINT16"></Variable>
61:     <Variable          name="amUpdateActivityExecStatusRcvd"          kind="tbd"
format="UINT16"></Variable>
62:     <Variable          name="amUpdateActivityExecStatusActd"          kind="tbd"
format="UINT16"></Variable>
63:     <Variable          name="amUpdateActivityExecStatusSucc"          kind="tbd"
format="UINT16"></Variable>
64:     <Variable name="amUpdateActivityExecStatusFail" kind="tbd" format="UINT16"></Variable>
65:     <Variable name="amDeleteActivityReceived" kind="tbd" format="UINT16"></Variable>
66:     <Variable name="amDeleteActivityAccepted" kind="tbd" format="UINT16"></Variable>
67:     <Variable name="amDeleteActivitySuccess" kind="tbd" format="UINT16"></Variable>
68:     <Variable name="amDeleteActivityFailure" kind="tbd" format="UINT16"></Variable>
69:     <Variable name="amDeleteAllActivitiesReceived" kind="tbd" format="UINT16"></Variable>
70:     <Variable name="amDeleteAllActivitiesAccepted" kind="tbd" format="UINT16"></Variable>
71:     <Variable name="amDeleteAllActivitiesSuccess" kind="tbd" format="UINT16"></Variable>
72:     <Variable name="amDeleteAllActivitiesFailure" kind="tbd" format="UINT16"></Variable>

```

```

73:     <Variable name="amPrepSchedFileReceived" kind="tbd" format="UINT16"></Variable>
74:     <Variable name="amPrepSchedFileAccepted" kind="tbd" format="UINT16"></Variable>
75:     <Variable name="amPrepSchedFileSuccess" kind="tbd" format="UINT16"></Variable>
76:     <Variable name="amPrepSchedFileFailure" kind="tbd" format="UINT16"></Variable>
77:     <Variable name="amSetDebugLevelReceived" kind="tbd" format="UINT16"></Variable>
78:     <Variable name="amSetDebugLevelAccepted" kind="tbd" format="UINT16"></Variable>
79:     <Variable name="amSetDebugLevelSuccess" kind="tbd" format="UINT16"></Variable>
80:     <Variable name="amSetDebugLevelFailure" kind="tbd" format="UINT16"></Variable>
81:
82:     <!-- Variables for resources required are tbd -->
83:
84:     <Request>
85:         <CommandMsg name="amSchedActivity" id="001" description="Schedule a mission or
housekeeping activity">
86:             <VariableRef name="amActivityId"/>
87:             <VariableRef name="amActivityType"/>
88:             <VariableRef name="amActivityPriority"/>
89:             <VariableRef name="amActivityBeginTime"/>
90:             <VariableRef name="amActivityDuration"/>
91:             <VariableRef name="amAllowConcurrentActivity"/>
92:             <VariableRef name="amActivityNotBeforeTime"/>
93:             <VariableRef name="amActivityNotAfterTime"/>
94:
95:             <!-- VariableRefs for resources are tbd -->
96:
97:         </CommandMsg>
98:         <DataReplyMsg name="amSchedActivityReply" id="002">
99:             <VariableRef name="amScheduledActivityId"/>
100:             <VariableRef name="amStatus"/>
101:         </DataReplyMsg>
102:     </Request>
103:
104:     <Request>
105:         <CommandMsg name="amAdjustActivityTime" id="003" description="Adjust the
scheduled time of a mission or housekeeping activity">
106:             <VariableRef name="amActivityId"/>
107:             <VariableRef name="amActivityBeginTime"/>
108:             <VariableRef name="amActivityDuration"/>
109:         </CommandMsg>
110:         <DataReplyMsg name="amAdjustActivityTimeReply" id="004">
111:             <VariableRef name="amActivityId"/>

```

```

112:         <VariableRef name="amStatus"/>
113:     </DataReplyMsg>
114: </Request>
115: <Request>
116:     <CommandMsg name="amGetSoh" id="011"></CommandMsg>
117:     <DataReplyMsg name="amGetSohReply" id="012">
118:         <VariableRef name="amActivitiesCurrentlyScheduled"/>
119:         <VariableRef name="amActivitiesExecuted"/>
120:         <VariableRef name="amActivitiesExecutedSuccess"/>
121:         <VariableRef name="amActivitiesExecutedFailed"/>
122:         <VariableRef name="amActivitiesDeleted"/>
123:         <VariableRef name="amSchedActivityReceived"/>
124:         <VariableRef name="amSchedActivityAccepted"/>
125:         <VariableRef name="amSchedActivitySuccess"/>
126:         <VariableRef name="amSchedActivityFailure"/>
127:         <VariableRef name="amAdjustActivityTimeReceived"/>
128:         <VariableRef name="amAdjustActivityTimeSuccess"/>
129:         <VariableRef name="amAdjustActivityTimeFailure"/>
130:         <VariableRef name="amExecuteActivityReceived"/>
131:         <VariableRef name="amExecuteActivityAccepted"/>
132:         <VariableRef name="amExecuteActivitySuccess"/>
133:         <VariableRef name="amExecuteActivityFailure"/>
134:         <VariableRef name="amUpdateActivityExecStatusRcvd"/>
135:         <VariableRef name="amUpdateActivityExecStatusActd"/>
136:         <VariableRef name="amUpdateActivityExecStatusSucc"/>
137:         <VariableRef name="amUpdateActivityExecStatusFail"/>
138:         <VariableRef name="amDeleteActivityReceived"/>
139:         <VariableRef name="amDeleteActivityAccepted"/>
140:         <VariableRef name="amDeleteActivitySuccess"/>
141:         <VariableRef name="amDeleteActivityFailure"/>
142:         <VariableRef name="amDeleteAllActivitiesReceived"/>
143:         <VariableRef name="amDeleteAllActivitiesAccepted"/>
144:         <VariableRef name="amDeleteAllActivitiesSuccess"/>
145:         <VariableRef name="amDeleteAllActivitiesFailure"/>
146:         <VariableRef name="amPrepSchedFileReceived"/>
147:         <VariableRef name="amPrepSchedFileAccepted"/>
148:         <VariableRef name="amPrepSchedFileSuccess"/>
149:         <VariableRef name="amPrepSchedFileFailure"/>
150:         <VariableRef name="amSetDebugLevelReceived"/>
151:         <VariableRef name="amSetDebugLevelAccepted"/>
152:         <VariableRef name="amSetDebugLevelSuccess"/>

```

```

153:         <VariableRef name="amSetDebugLevelFailure"/>
154:         <VariableRef name="amCurrentDebugLevel"/>
155:     </DataReplyMsg>
156: </Request>
157:
158: <Notification>
159:     <DataMsg name="amSOH" id="013" msgArrival="EVENT">
160:         <Qualifier value="1" name="telemetryLevel"></Qualifier>
161:         <VariableRef name="amActivitiesCurrentlyScheduled"/>
162:         <VariableRef name="amActivitiesExecuted"/>
163:         <VariableRef name="amActivitiesExecutedSuccess"/>
164:         <VariableRef name="amActivitiesExecutedFailed"/>
165:         <VariableRef name="amActivitiesDeleted"/>
166:         <VariableRef name="amSchedActivityReceived"/>
167:         <VariableRef name="amSchedActivityAccepted"/>
168:         <VariableRef name="amSchedActivitySuccess"/>
169:         <VariableRef name="amSchedActivityFailure"/>
170:         <VariableRef name="amAdjustActivityTimeReceived"/>
171:         <VariableRef name="amAdjustActivityTimeSuccess"/>
172:         <VariableRef name="amAdjustActivityTimeFailure"/>
173:         <VariableRef name="amExecuteActivityReceived"/>
174:         <VariableRef name="amExecuteActivityAccepted"/>
175:         <VariableRef name="amExecuteActivitySuccess"/>
176:         <VariableRef name="amExecuteActivityFailure"/>
177:         <VariableRef name="amUpdateActivityExecStatusRcvd"/>
178:         <VariableRef name="amUpdateActivityExecStatusActd"/>
179:         <VariableRef name="amUpdateActivityExecStatusSucc"/>
180:         <VariableRef name="amUpdateActivityExecStatusFail"/>
181:         <VariableRef name="amDeleteActivityReceived"/>
182:         <VariableRef name="amDeleteActivityAccepted"/>
183:         <VariableRef name="amDeleteActivitySuccess"/>
184:         <VariableRef name="amDeleteActivityFailure"/>
185:         <VariableRef name="amDeleteAllActivitiesReceived"/>
186:         <VariableRef name="amDeleteAllActivitiesAccepted"/>
187:         <VariableRef name="amDeleteAllActivitiesSuccess"/>
188:         <VariableRef name="amDeleteAllActivitiesFailure"/>
189:         <VariableRef name="amPrepSchedFileReceived"/>
190:         <VariableRef name="amPrepSchedFileAccepted"/>
191:         <VariableRef name="amPrepSchedFileSuccess"/>
192:         <VariableRef name="amPrepSchedFileFailure"/>
193:         <VariableRef name="amSetDebugLevelReceived"/>

```

```

194:         <VariableRef name="amSetDebugLevelAccepted"/>
195:         <VariableRef name="amSetDebugLevelSuccess"/>
196:         <VariableRef name="amSetDebugLevelFailure"/>
197:         <VariableRef name="amCurrentDebugLevel"/>
198:     </DataMsg>
199: </Notification>
200:
201: </Interface>
202: <!-- Component Safety Interface, defining and facilitating monitoring of device health and safety
-->
203: <Interface name="CmpSafety" id="10">
204:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
205:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
206:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
207:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
208:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
209:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds"/>
210:     <Variable name="DeviceTemperature" kind="temperature" format="INT16"
scaleFactor="1.0" scaleUnits="degC"/>
211:     <Variable name="PanelTemperatureArray" length="8" kind="temperature" format="INT16"
scaleFactor="1.0" scaleUnits="degC">
212:         <Qualifier name="PanelTemperatureChannel" value="Array_8"/>
213:     </Variable>
214:
215:     <Notification>
216:         <DataMsg name="DeviceTemp" id="1" msgArrival="PERIODIC" msgRate="1">
217:             <Qualifier name="telemetryLevel" value="1"/>
218:             <VariableRef name="Time"/>
219:             <VariableRef name="SubS"/>
220:             <VariableRef name="DeviceTemperature"/>
221:             <VariableRef name="PanelTemperatureArray"/>
222:         </DataMsg>
223:     </Notification>
224:
225: </Interface>
226: </xTEDS>
227:

```


File: sdm/app/test/DMTTests/LargexTEDSSmaller.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd"
name="ActivityManagerXTEDS" description="ActivityManager xTEDS" version="1.0">
4:
5: <Application name="ActivityManager" kind="AutonomyFlightSoftware" description="Autonomous
Tasking Executive (ATE), ActivityManager"/>
6:
7: <Interface name="ActivityManagerInterface" id="1">
8:
9:     <Variable name="amActivityId" kind="ID" format="UINT32"></Variable>
10:    <Variable name="amActivityType" kind="tbd" format="UINT08" length="33"></Variable><!--
33-byte, null terminated string -->
11:    <Variable name="amActivityPriority" kind="tbd" format="UINT16">
12:        <Drange name="amActivityPriorityEnum">
13:            <Option value="0" name="LOW"/>
14:            <Option value="1" name="MEDIUM"/>
15:            <Option value="2" name="HIGH"/>
16:        </Drange>
17:    </Variable>
18:    <Variable      name="amActivityBeginTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies as soon as possible -->
19:    <Variable name="amActivityDuration" kind="Time" format="FLOAT64" units="s"></Variable>
20:    <Variable      name="amActivityNotBeforeTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies now -->
21:    <Variable      name="amActivityNotAfterTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies no limit -->
22:    <Variable name="amAllowConcurrentActivity" kind="boolean" format="UINT08">
23:        <Drange name="amAllowConcurrentActivityEnum">
24:            <Option value="0" name="NO"/>
25:            <Option value="1" name="YES"/>
26:        </Drange>
27:    </Variable>
28:    <Variable name="amScheduledActivityId" kind="ID" format="UINT32"></Variable>
29:    <Variable name="amStatus" kind="Status" format="INT16">
30:        <Drange name="amStatusEnum">
31:            <Option value="0" name="FAILURE"/>
32:            <Option value="1" name="SUCCESS"/>
33:        </Drange>
```

```

34:     </Variable>
35:     <Variable name="amExecStatus" kind="Status" format="INT16">
36:         <Drange name="amExecStatusEnum">
37:             <Option value="0" name="FAILURE"/>
38:             <Option value="1" name="SUCCESS"/>
39:             <Option value="2" name="NOT_EXECUTED"/>
40:         </Drange>
41:     </Variable>
42:     <Variable name="amDebugLevel" kind="tbd" format="UINT16"></Variable>
43:     <Variable name="amCurrentDebugLevel" kind="tbd" format="UINT16"></Variable>
44:     <Variable name="amActivitiesCurrentlyScheduled" kind="tbd" format="UINT16"></Variable>
45:     <Variable name="amActivitiesExecuted" kind="tbd" format="UINT16"></Variable>
46:     <Variable name="amActivitiesExecutedSuccess" kind="tbd" format="UINT16"></Variable>
47:     <Variable name="amActivitiesExecutedFailed" kind="tbd" format="UINT16"></Variable>
48:     <Variable name="amActivitiesDeleted" kind="tbd" format="UINT16"></Variable>
49:     <Variable name="amSchedActivityReceived" kind="tbd" format="UINT16"></Variable>
50:     <Variable name="amSchedActivityAccepted" kind="tbd" format="UINT16"></Variable>
51:     <Variable name="amSchedActivitySuccess" kind="tbd" format="UINT16"></Variable>
52:     <Variable name="amSchedActivityFailure" kind="tbd" format="UINT16"></Variable>
53:     <Variable name="amAdjustActivityTimeReceived" kind="tbd" format="UINT16"></Variable>
54:     <Variable name="amAdjustActivityTimeAccepted" kind="tbd" format="UINT16"></Variable>
55:     <Variable name="amAdjustActivityTimeSuccess" kind="tbd" format="UINT16"></Variable>
56:     <Variable name="amAdjustActivityTimeFailure" kind="tbd" format="UINT16"></Variable>
57:     <Variable name="amExecuteActivityReceived" kind="tbd" format="UINT16"></Variable>
58:     <Variable name="amExecuteActivityAccepted" kind="tbd" format="UINT16"></Variable>
59:     <Variable name="amExecuteActivitySuccess" kind="tbd" format="UINT16"></Variable>
60:     <Variable name="amExecuteActivityFailure" kind="tbd" format="UINT16"></Variable>
61:     <Variable          name="amUpdateActivityExecStatusRcvd"          kind="tbd"
format="UINT16"></Variable>
62:     <Variable          name="amUpdateActivityExecStatusActd"          kind="tbd"
format="UINT16"></Variable>
63:     <Variable          name="amUpdateActivityExecStatusSucc"          kind="tbd"
format="UINT16"></Variable>
64:     <Variable name="amUpdateActivityExecStatusFail" kind="tbd" format="UINT16"></Variable>
65:     <Variable name="amDeleteActivityReceived" kind="tbd" format="UINT16"></Variable>
66:     <Variable name="amDeleteActivityAccepted" kind="tbd" format="UINT16"></Variable>
67:     <Variable name="amDeleteActivitySuccess" kind="tbd" format="UINT16"></Variable>
68:     <Variable name="amDeleteActivityFailure" kind="tbd" format="UINT16"></Variable>
69:     <Variable name="amDeleteAllActivitiesReceived" kind="tbd" format="UINT16"></Variable>
70:     <Variable name="amDeleteAllActivitiesAccepted" kind="tbd" format="UINT16"></Variable>
71:     <Variable name="amDeleteAllActivitiesSuccess" kind="tbd" format="UINT16"></Variable>
72:     <Variable name="amDeleteAllActivitiesFailure" kind="tbd" format="UINT16"></Variable>

```

```

73:     <Variable name="amPrepSchedFileReceived" kind="tbd" format="UINT16"></Variable>
74:     <Variable name="amPrepSchedFileAccepted" kind="tbd" format="UINT16"></Variable>
75:     <Variable name="amPrepSchedFileSuccess" kind="tbd" format="UINT16"></Variable>
76:     <Variable name="amPrepSchedFileFailure" kind="tbd" format="UINT16"></Variable>
77:     <Variable name="amSetDebugLevelReceived" kind="tbd" format="UINT16"></Variable>
78:     <Variable name="amSetDebugLevelAccepted" kind="tbd" format="UINT16"></Variable>
79:     <Variable name="amSetDebugLevelSuccess" kind="tbd" format="UINT16"></Variable>
80:     <Variable name="amSetDebugLevelFailure" kind="tbd" format="UINT16"></Variable>
81:
82:     <!-- Variables for resources required are tbd -->
83:
84:     <Request>
85:         <CommandMsg name="amSchedActivity" id="001" description="Schedule a mission or
housekeeping activity">
86:             <VariableRef name="amActivityId"/>
87:             <VariableRef name="amActivityType"/>
88:             <VariableRef name="amActivityPriority"/>
89:             <VariableRef name="amActivityBeginTime"/>
90:             <VariableRef name="amActivityDuration"/>
91:             <VariableRef name="amAllowConcurrentActivity"/>
92:             <VariableRef name="amActivityNotBeforeTime"/>
93:             <VariableRef name="amActivityNotAfterTime"/>
94:
95:             <!-- VariableRefs for resources are tbd -->
96:
97:         </CommandMsg>
98:         <DataReplyMsg name="amSchedActivityReply" id="002">
99:             <VariableRef name="amScheduledActivityId"/>
100:             <VariableRef name="amStatus"/>
101:         </DataReplyMsg>
102:     </Request>
103:
104:     <Request>
105:         <CommandMsg name="amAdjustActivityTime" id="003" description="Adjust the
scheduled time of a mission or housekeeping activity">
106:             <VariableRef name="amActivityId"/>
107:             <VariableRef name="amActivityBeginTime"/>
108:             <VariableRef name="amActivityDuration"/>
109:         </CommandMsg>
110:         <DataReplyMsg name="amAdjustActivityTimeReply" id="004">
111:             <VariableRef name="amActivityId"/>

```

```

112:         <VariableRef name="amStatus"/>
113:     </DataReplyMsg>
114: </Request>
115:
116: <Command>
117:     <CommandMsg name="amExecuteActivity" id="005" description="Execute a scheduled
mission or housekeeping activity">
118:         <VariableRef name="amActivityId"/>
119:     </CommandMsg>
120: </Command>
121:
122: <Command>
123:     <CommandMsg name="amUpdateActivityExecStatus" id="006" description="Update
activity execution statistics">
124:         <VariableRef name="amActivityId"/>
125:         <VariableRef name="amExecStatus"/>
126:     </CommandMsg>
127: </Command>
128:
129: <Command>
130:     <CommandMsg name="amDeleteActivity" id="007" description="Delete a scheduled
activity">
131:         <VariableRef name="amActivityId"/>
132:     </CommandMsg>
133: </Command>
134:
135: <Command>
136:     <CommandMsg name="amDeleteAllActivities" id="008" description="Delete all
scheduled activities"></CommandMsg>
137: </Command>
138:
139: <Command>
140:     <CommandMsg name="amPrepSchedFile" id="009" description="Prepare an activities
schedule for downlink"></CommandMsg>
141: </Command>
142:
143: <Command>
144:     <CommandMsg name="amSetDebugLevel" id="010" description="Set the debug log
verbosity level">
145:         <VariableRef name="amDebugLevel"/>
146:     </CommandMsg>
147: </Command>

```

```

148:
149:     <Request>
150:         <CommandMsg name="amGetSoh" id="011"></CommandMsg>
151:         <DataReplyMsg name="amGetSohReply" id="012">
152:             <VariableRef name="amActivitiesCurrentlyScheduled"/>
153:             <VariableRef name="amActivitiesExecuted"/>
154:             <VariableRef name="amActivitiesExecutedSuccess"/>
155:             <VariableRef name="amActivitiesExecutedFailed"/>
156:             <VariableRef name="amActivitiesDeleted"/>
157:             <VariableRef name="amSchedActivityReceived"/>
158:             <VariableRef name="amSchedActivityAccepted"/>
159:             <VariableRef name="amSchedActivitySuccess"/>
160:             <VariableRef name="amSchedActivityFailure"/>
161:             <VariableRef name="amAdjustActivityTimeReceived"/>
162:             <VariableRef name="amAdjustActivityTimeSuccess"/>
163:             <VariableRef name="amAdjustActivityTimeFailure"/>
164:             <VariableRef name="amExecuteActivityReceived"/>
165:             <VariableRef name="amExecuteActivityAccepted"/>
166:             <VariableRef name="amExecuteActivitySuccess"/>
167:             <VariableRef name="amExecuteActivityFailure"/>
168:             <VariableRef name="amUpdateActivityExecStatusRcvd"/>
169:             <VariableRef name="amUpdateActivityExecStatusActd"/>
170:             <VariableRef name="amUpdateActivityExecStatusSucc"/>
171:             <VariableRef name="amUpdateActivityExecStatusFail"/>
172:             <VariableRef name="amDeleteActivityReceived"/>
173:             <VariableRef name="amDeleteActivityAccepted"/>
174:             <VariableRef name="amDeleteActivitySuccess"/>
175:             <VariableRef name="amDeleteActivityFailure"/>
176:             <VariableRef name="amDeleteAllActivitiesReceived"/>
177:             <VariableRef name="amDeleteAllActivitiesAccepted"/>
178:             <VariableRef name="amDeleteAllActivitiesSuccess"/>
179:             <VariableRef name="amDeleteAllActivitiesFailure"/>
180:             <VariableRef name="amPrepSchedFileReceived"/>
181:             <VariableRef name="amPrepSchedFileAccepted"/>
182:             <VariableRef name="amPrepSchedFileSuccess"/>
183:             <VariableRef name="amPrepSchedFileFailure"/>
184:             <VariableRef name="amSetDebugLevelReceived"/>
185:             <VariableRef name="amSetDebugLevelAccepted"/>
186:             <VariableRef name="amSetDebugLevelSuccess"/>
187:             <VariableRef name="amSetDebugLevelFailure"/>
188:             <VariableRef name="amCurrentDebugLevel"/>

```

```

189:         </DataReplyMsg>
190:     </Request>
191:
192:     <Notification>
193:         <DataMsg name="amSOH" id="013" msgArrival="EVENT">
194:             <Qualifier value="1" name="telemetryLevel"></Qualifier>
195:             <VariableRef name="amActivitiesCurrentlyScheduled"/>
196:             <VariableRef name="amActivitiesExecuted"/>
197:             <VariableRef name="amActivitiesExecutedSuccess"/>
198:             <VariableRef name="amActivitiesExecutedFailed"/>
199:             <VariableRef name="amActivitiesDeleted"/>
200:             <VariableRef name="amSchedActivityReceived"/>
201:             <VariableRef name="amSchedActivityAccepted"/>
202:             <VariableRef name="amSchedActivitySuccess"/>
203:             <VariableRef name="amSchedActivityFailure"/>
204:             <VariableRef name="amAdjustActivityTimeReceived"/>
205:             <VariableRef name="amAdjustActivityTimeSuccess"/>
206:             <VariableRef name="amAdjustActivityTimeFailure"/>
207:             <VariableRef name="amExecuteActivityReceived"/>
208:             <VariableRef name="amExecuteActivityAccepted"/>
209:             <VariableRef name="amExecuteActivitySuccess"/>
210:             <VariableRef name="amExecuteActivityFailure"/>
211:             <VariableRef name="amUpdateActivityExecStatusRcvd"/>
212:             <VariableRef name="amUpdateActivityExecStatusActd"/>
213:             <VariableRef name="amUpdateActivityExecStatusSucc"/>
214:             <VariableRef name="amUpdateActivityExecStatusFail"/>
215:             <VariableRef name="amDeleteActivityReceived"/>
216:             <VariableRef name="amDeleteActivityAccepted"/>
217:             <VariableRef name="amDeleteActivitySuccess"/>
218:             <VariableRef name="amDeleteActivityFailure"/>
219:             <VariableRef name="amDeleteAllActivitiesReceived"/>
220:             <VariableRef name="amDeleteAllActivitiesAccepted"/>
221:             <VariableRef name="amDeleteAllActivitiesSuccess"/>
222:             <VariableRef name="amDeleteAllActivitiesFailure"/>
223:             <VariableRef name="amPrepSchedFileReceived"/>
224:             <VariableRef name="amPrepSchedFileAccepted"/>
225:             <VariableRef name="amPrepSchedFileSuccess"/>
226:             <VariableRef name="amPrepSchedFileFailure"/>
227:             <VariableRef name="amSetDebugLevelReceived"/>
228:             <VariableRef name="amSetDebugLevelAccepted"/>
229:             <VariableRef name="amSetDebugLevelSuccess"/>

```

```

230:         <VariableRef name="amSetDebugLevelFailure"/>
231:         <VariableRef name="amCurrentDebugLevel"/>
232:     </DataMsg>
233: </Notification>
234:
235: </Interface>
236: <!-- Panel interface, encapsulating "panel-level" power housekeeping -->
237: <Interface id="8" name="PanelPowerInterface" description="Panel Power Interface">
238:     <Qualifier name="NumberOfPorts" value="4"/>
239:     <Qualifier name="BreakerTripCurrent" value="30.0" units="Amps"/>
240:     <Variable name="PortReference" kind="PowerPortNumber" format="UINT08"/>
241:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
242:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds"/>
243:     <Variable name="PanelVoltage" kind="Voltage" format="INT16" scaleFactor="0.1"
scaleUnits="Volts"/>
244:     <Variable name="BreakerCurrentArray" length="4" kind="Current" format="INT16"
scaleFactor="1.0" scaleUnits="Amps">
245:         <Qualifier name="Panel" value="Array_4"/>
246:     </Variable>
247:     <Variable name="PortPowerStateArray" length="4" kind="enumeration"
format="UINT08">
248:         <Drange name="PowerStateEnum">
249:             <Option name="Open" value="0"/>
250:             <Option name="Closed" value="1"/>
251:         </Drange>
252:     </Variable>
253:
254: <!-- Message definitions -->
255: <Notification>
256:     <DataMsg id="1" name="PowerStatus" msgArrival="PERIODIC" msgRate="1" >
257:         <Qualifier name="telemetryLevel" value="1"/>
258:         <VariableRef name="Time"/>
259:         <VariableRef name="SubS"/>
260:         <VariableRef name="PanelVoltage"/>
261:         <VariableRef name="BreakerCurrentArray"/>
262:         <VariableRef name="PortPowerStateArray"/>
263:     </DataMsg>
264: </Notification>
265:
266: <!-- Commands and services -->
267: <Command>

```

```

268:         <CommandMsg id="2" name="PortPowerOn" >
269:             <VariableRef name="PortReference"/>
270:         </CommandMsg>
271:     </Command>
272:
273:     <Command>
274:         <CommandMsg id="3" name="PortPowerOff" >
275:             <VariableRef name="PortReference"/>
276:         </CommandMsg>
277:     </Command>
278:
279: </Interface>
280:
281:
282: <Interface id="9" name="RoboHubInterface" description="Interface for a robust hub">
283:     <!-- Properties of the hub and variables in "hub" scope, not "port" scope -->
284:     <Qualifier name="NumberOfPorts" value="8"/>
285:     <Qualifier name="PortTripCurrent" value="4.5" units="Amps"/>
286:     <Variable      name="PortReference"      kind="HubPortNumber"      format="UINT08"
description="Refers to a port for events and commands"/>
287:     <Variable  name="PortEnumeration"  kind="HubEnumerationStatus"  format="UINT08"
description="Used to indicate hub enumeration state"/>
288:     <Variable  name="PPS_Status"  kind="PpsStatus"  format="UINT08"  description="Used to
indicate PPS status"/>
289:     <Variable      name="SetTripCurrentVal"      kind="TripCurrent"      format="INT16"
scaleFactor="0.1" defaultValue="10" scaleUnits="Amps"/>
290:     <Variable name="PortPowerState" kind="PortPowerState" format="UINT08">
291:         <Drange name="PowerStateEnum">
292:             <Option name="Open" value="0"/>
293:             <Option name="Closed" value="1"/>
294:             <Option name="HardTrip" value="2"/>
295:             <Option name="SoftTrip" value="3"/>
296:         </Drange>
297:     </Variable>
298:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
299:     <Variable  kind="SubSeconds"  name="SubS"  units="Counts"  format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
300:
301:     <!-- Variables in "port" scope, declared for each port as arrays of length N_Ports -->
302:     <Variable  name="PortVoltageArray"  length="8"  kind="Voltage"  format="INT16"
scaleFactor="0.1" scaleUnits="Volts">
303:         <Qualifier name="HubPort" value="Array_8"/>

```



```

304:     </Variable>
305:     <Variable name="PortCurrentArray" length="8" kind="Current" format="INT16"
scaleFactor="0.1" scaleUnits="Amps">
306:         <Qualifier name="HubPort" value="Array_8"/>
307:     </Variable>
308:     <Variable name="SoftCurrentLimitArray" length="8" kind="TripCurrent" format="INT16"
scaleFactor="0.1" defaultValue="10" scaleUnits="Amps">
309:         <Qualifier name="HubPort" value="Array_8"/>
310:     </Variable>
311:     <Variable name="PortPowerStateArray" length="8" kind="boolean" format="UINT08">
312:         <Qualifier name="HubPort" value="Array_8"/>
313:         <Drange name="PowerStateEnumArray">
314:             <Option name="Open" value="0"/>
315:             <Option name="Closed" value="1"/>
316:             <Option name="HardTrip" value="2"/>
317:             <Option name="SoftTrip" value="3"/>
318:         </Drange>
319:     </Variable>
320:
321:     <!-- Message Definitions -->
322:     <Notification>
323:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1">
324:             <Qualifier name="telemetryLevel" value="1"/>
325:             <VariableRef name="Time"/>
326:             <VariableRef name="SubS"/>
327:             <VariableRef name="PortPowerStateArray"/>
328:             <VariableRef name="PortVoltageArray"/>
329:             <VariableRef name="PortCurrentArray"/>
330:         </DataMsg>
331:     </Notification>
332:
333:     <Notification>
334:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
335:             <Qualifier name="telemetryLevel" value="1"/>
336:             <VariableRef name="Time"/>
337:             <VariableRef name="SubS"/>
338:             <VariableRef name="PortReference"/>
339:             <VariableRef name="PortPowerState"/>
340:         </DataMsg>
341:     </Notification>
342:

```

```

343:     <!-- Commands and services -->
344:     <Command>
345:         <CommandMsg id="3" name="ConfigureSoftTrip">
346:             <VariableRef name="PortReference"/>
347:             <VariableRef name="SetTripCurrentVal"/>
348:         </CommandMsg>
349:     </Command>
350:
351:     <Command>
352:         <CommandMsg id="4" name="PortPowerOn" >
353:             <VariableRef name="PortReference"/>
354:         </CommandMsg>
355:     </Command>
356:     <Command>
357:         <CommandMsg id="5" name="PortPowerOff" >
358:             <VariableRef name="PortReference"/>
359:         </CommandMsg>
360:     </Command>
361:
362:     <Request>
363:         <CommandMsg id="6" name="GetHubStatus"/>
364:         <DataReplyMsg id="7" name="HubStatusReply">
365:             <VariableRef name="Time"/>
366:             <VariableRef name="SubS"/>
367:             <VariableRef name="PortEnumeration"/>
368:             <VariableRef name="PPS_Status"/>
369:         </DataReplyMsg>
370:     </Request>
371:
372:     <Request>
373:         <CommandMsg id="8" name="GetSoftTripLimits"/>
374:         <DataReplyMsg id="9" name="SoftLimitSettingsReply">
375:             <VariableRef name="Time"/>
376:             <VariableRef name="SubS"/>
377:             <VariableRef name="SoftCurrentLimitArray"/>
378:         </DataReplyMsg>
379:     </Request>
380:
381: </Interface>
382: <!-- Component Safety Interface, defining and facilitating monitoring of device health and safety
-->

```

```

383:   <Interface name="CmpSafety" id="10">
384:       <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
385:       <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
386:       <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
387:       <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
388:       <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
389:       <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds"/>
390:       <Variable name="DeviceTemperature" kind="temperature" format="INT16"
scaleFactor="1.0" scaleUnits="degC"/>
391:       <Variable name="PanelTemperatureArray" length="8" kind="temperature" format="INT16"
scaleFactor="1.0" scaleUnits="degC">
392:           <Qualifier name="PanelTemperatureChannel" value="Array_8"/>
393:       </Variable>
394:
395:       <Notification>
396:           <DataMsg name="DeviceTemp" id="1" msgArrival="PERIODIC" msgRate="1">
397:               <Qualifier name="telemetryLevel" value="1"/>
398:               <VariableRef name="Time"/>
399:               <VariableRef name="SubS"/>
400:               <VariableRef name="DeviceTemperature"/>
401:               <VariableRef name="PanelTemperatureArray"/>
402:           </DataMsg>
403:       </Notification>
404:
405:   </Interface>
406: </xTEDS>
407:

```

File: sdm/app/test/DMTests/LargexTEDS.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd"
name="ActivityManagerXTEDS" description="ActivityManager xTEDS" version="1.0">
4:
5:   <Application name="ActivityManager" kind="AutonomyFlightSoftware" description="Autonomous
Tasking Executive (ATE), ActivityManager"/>
6:
7:   <Interface name="ActivityManagerInterface" id="1">
8:
9:     <Variable name="amActivityId" kind="ID" format="UINT32"></Variable>
10:    <Variable name="amActivityType" kind="tbd" format="UINT08" length="33"></Variable><!--
33-byte, null terminated string -->
11:    <Variable name="amActivityPriority" kind="tbd" format="UINT16">
12:      <Drange name="amActivityPriorityEnum">
13:        <Option value="0" name="LOW"/>
14:        <Option value="1" name="MEDIUM"/>
15:        <Option value="2" name="HIGH"/>
16:      </Drange>
17:    </Variable>
18:    <Variable      name="amActivityBeginTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies as soon as possible -->
19:    <Variable name="amActivityDuration" kind="Time" format="FLOAT64" units="s"></Variable>
20:    <Variable      name="amActivityNotBeforeTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies now -->
21:    <Variable      name="amActivityNotAfterTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies no limit -->
22:    <Variable name="amAllowConcurrentActivity" kind="boolean" format="UINT08">
23:      <Drange name="amAllowConcurrentActivityEnum">
24:        <Option value="0" name="NO"/>
25:        <Option value="1" name="YES"/>
26:      </Drange>
27:    </Variable>
28:    <Variable name="amScheduledActivityId" kind="ID" format="UINT32"></Variable>
29:    <Variable name="amStatus" kind="Status" format="INT16">
30:      <Drange name="amStatusEnum">
31:        <Option value="0" name="FAILURE"/>
32:        <Option value="1" name="SUCCESS"/>
33:      </Drange>
```

```

34:     </Variable>
35:     <Variable name="amExecStatus" kind="Status" format="INT16">
36:         <Drange name="amExecStatusEnum">
37:             <Option value="0" name="FAILURE"/>
38:             <Option value="1" name="SUCCESS"/>
39:             <Option value="2" name="NOT_EXECUTED"/>
40:         </Drange>
41:     </Variable>
42:     <Variable name="amDebugLevel" kind="tbd" format="UINT16"></Variable>
43:     <Variable name="amCurrentDebugLevel" kind="tbd" format="UINT16"></Variable>
44:     <Variable name="amActivitiesCurrentlyScheduled" kind="tbd" format="UINT16"></Variable>
45:     <Variable name="amActivitiesExecuted" kind="tbd" format="UINT16"></Variable>
46:     <Variable name="amActivitiesExecutedSuccess" kind="tbd" format="UINT16"></Variable>
47:     <Variable name="amActivitiesExecutedFailed" kind="tbd" format="UINT16"></Variable>
48:     <Variable name="amActivitiesDeleted" kind="tbd" format="UINT16"></Variable>
49:     <Variable name="amSchedActivityReceived" kind="tbd" format="UINT16"></Variable>
50:     <Variable name="amSchedActivityAccepted" kind="tbd" format="UINT16"></Variable>
51:     <Variable name="amSchedActivitySuccess" kind="tbd" format="UINT16"></Variable>
52:     <Variable name="amSchedActivityFailure" kind="tbd" format="UINT16"></Variable>
53:     <Variable name="amAdjustActivityTimeReceived" kind="tbd" format="UINT16"></Variable>
54:     <Variable name="amAdjustActivityTimeAccepted" kind="tbd" format="UINT16"></Variable>
55:     <Variable name="amAdjustActivityTimeSuccess" kind="tbd" format="UINT16"></Variable>
56:     <Variable name="amAdjustActivityTimeFailure" kind="tbd" format="UINT16"></Variable>
57:     <Variable name="amExecuteActivityReceived" kind="tbd" format="UINT16"></Variable>
58:     <Variable name="amExecuteActivityAccepted" kind="tbd" format="UINT16"></Variable>
59:     <Variable name="amExecuteActivitySuccess" kind="tbd" format="UINT16"></Variable>
60:     <Variable name="amExecuteActivityFailure" kind="tbd" format="UINT16"></Variable>
61:     <Variable          name="amUpdateActivityExecStatusRcvd"          kind="tbd"
format="UINT16"></Variable>
62:     <Variable          name="amUpdateActivityExecStatusActd"          kind="tbd"
format="UINT16"></Variable>
63:     <Variable          name="amUpdateActivityExecStatusSucc"          kind="tbd"
format="UINT16"></Variable>
64:     <Variable name="amUpdateActivityExecStatusFail" kind="tbd" format="UINT16"></Variable>
65:     <Variable name="amDeleteActivityReceived" kind="tbd" format="UINT16"></Variable>
66:     <Variable name="amDeleteActivityAccepted" kind="tbd" format="UINT16"></Variable>
67:     <Variable name="amDeleteActivitySuccess" kind="tbd" format="UINT16"></Variable>
68:     <Variable name="amDeleteActivityFailure" kind="tbd" format="UINT16"></Variable>
69:     <Variable name="amDeleteAllActivitiesReceived" kind="tbd" format="UINT16"></Variable>
70:     <Variable name="amDeleteAllActivitiesAccepted" kind="tbd" format="UINT16"></Variable>
71:     <Variable name="amDeleteAllActivitiesSuccess" kind="tbd" format="UINT16"></Variable>
72:     <Variable name="amDeleteAllActivitiesFailure" kind="tbd" format="UINT16"></Variable>

```

```

73:     <Variable name="amPrepSchedFileReceived" kind="tbd" format="UINT16"></Variable>
74:     <Variable name="amPrepSchedFileAccepted" kind="tbd" format="UINT16"></Variable>
75:     <Variable name="amPrepSchedFileSuccess" kind="tbd" format="UINT16"></Variable>
76:     <Variable name="amPrepSchedFileFailure" kind="tbd" format="UINT16"></Variable>
77:     <Variable name="amSetDebugLevelReceived" kind="tbd" format="UINT16"></Variable>
78:     <Variable name="amSetDebugLevelAccepted" kind="tbd" format="UINT16"></Variable>
79:     <Variable name="amSetDebugLevelSuccess" kind="tbd" format="UINT16"></Variable>
80:     <Variable name="amSetDebugLevelFailure" kind="tbd" format="UINT16"></Variable>
81:
82:     <!-- Variables for resources required are tbd -->
83:
84:     <Request>
85:         <CommandMsg name="amSchedActivity" id="001" description="Schedule a mission or
housekeeping activity">
86:             <VariableRef name="amActivityId"/>
87:             <VariableRef name="amActivityType"/>
88:             <VariableRef name="amActivityPriority"/>
89:             <VariableRef name="amActivityBeginTime"/>
90:             <VariableRef name="amActivityDuration"/>
91:             <VariableRef name="amAllowConcurrentActivity"/>
92:             <VariableRef name="amActivityNotBeforeTime"/>
93:             <VariableRef name="amActivityNotAfterTime"/>
94:
95:             <!-- VariableRefs for resources are tbd -->
96:
97:         </CommandMsg>
98:         <DataReplyMsg name="amSchedActivityReply" id="002">
99:             <VariableRef name="amScheduledActivityId"/>
100:             <VariableRef name="amStatus"/>
101:         </DataReplyMsg>
102:     </Request>
103:
104:     <Request>
105:         <CommandMsg name="amAdjustActivityTime" id="003" description="Adjust the
scheduled time of a mission or housekeeping activity">
106:             <VariableRef name="amActivityId"/>
107:             <VariableRef name="amActivityBeginTime"/>
108:             <VariableRef name="amActivityDuration"/>
109:         </CommandMsg>
110:         <DataReplyMsg name="amAdjustActivityTimeReply" id="004">
111:             <VariableRef name="amActivityId"/>

```

```

112:         <VariableRef name="amStatus"/>
113:     </DataReplyMsg>
114: </Request>
115:
116: <Command>
117:     <CommandMsg name="amExecuteActivity" id="005" description="Execute a scheduled
mission or housekeeping activity">
118:         <VariableRef name="amActivityId"/>
119:     </CommandMsg>
120: </Command>
121:
122: <Command>
123:     <CommandMsg name="amUpdateActivityExecStatus" id="006" description="Update
activity execution statistics">
124:         <VariableRef name="amActivityId"/>
125:         <VariableRef name="amExecStatus"/>
126:     </CommandMsg>
127: </Command>
128:
129: <Command>
130:     <CommandMsg name="amDeleteActivity" id="007" description="Delete a scheduled
activity">
131:         <VariableRef name="amActivityId"/>
132:     </CommandMsg>
133: </Command>
134:
135: <Command>
136:     <CommandMsg name="amDeleteAllActivities" id="008" description="Delete all
scheduled activities"></CommandMsg>
137: </Command>
138:
139: <Command>
140:     <CommandMsg name="amPrepSchedFile" id="009" description="Prepare an activities
schedule for downlink"></CommandMsg>
141: </Command>
142:
143: <Command>
144:     <CommandMsg name="amSetDebugLevel" id="010" description="Set the debug log
verbosity level">
145:         <VariableRef name="amDebugLevel"/>
146:     </CommandMsg>
147: </Command>

```

```

148:
149:     <Request>
150:         <CommandMsg name="amGetSoh" id="011"></CommandMsg>
151:         <DataReplyMsg name="amGetSohReply" id="012">
152:             <VariableRef name="amActivitiesCurrentlyScheduled"/>
153:             <VariableRef name="amActivitiesExecuted"/>
154:             <VariableRef name="amActivitiesExecutedSuccess"/>
155:             <VariableRef name="amActivitiesExecutedFailed"/>
156:             <VariableRef name="amActivitiesDeleted"/>
157:             <VariableRef name="amSchedActivityReceived"/>
158:             <VariableRef name="amSchedActivityAccepted"/>
159:             <VariableRef name="amSchedActivitySuccess"/>
160:             <VariableRef name="amSchedActivityFailure"/>
161:             <VariableRef name="amAdjustActivityTimeReceived"/>
162:             <VariableRef name="amAdjustActivityTimeSuccess"/>
163:             <VariableRef name="amAdjustActivityTimeFailure"/>
164:             <VariableRef name="amExecuteActivityReceived"/>
165:             <VariableRef name="amExecuteActivityAccepted"/>
166:             <VariableRef name="amExecuteActivitySuccess"/>
167:             <VariableRef name="amExecuteActivityFailure"/>
168:             <VariableRef name="amUpdateActivityExecStatusRcvd"/>
169:             <VariableRef name="amUpdateActivityExecStatusActd"/>
170:             <VariableRef name="amUpdateActivityExecStatusSucc"/>
171:             <VariableRef name="amUpdateActivityExecStatusFail"/>
172:             <VariableRef name="amDeleteActivityReceived"/>
173:             <VariableRef name="amDeleteActivityAccepted"/>
174:             <VariableRef name="amDeleteActivitySuccess"/>
175:             <VariableRef name="amDeleteActivityFailure"/>
176:             <VariableRef name="amDeleteAllActivitiesReceived"/>
177:             <VariableRef name="amDeleteAllActivitiesAccepted"/>
178:             <VariableRef name="amDeleteAllActivitiesSuccess"/>
179:             <VariableRef name="amDeleteAllActivitiesFailure"/>
180:             <VariableRef name="amPrepSchedFileReceived"/>
181:             <VariableRef name="amPrepSchedFileAccepted"/>
182:             <VariableRef name="amPrepSchedFileSuccess"/>
183:             <VariableRef name="amPrepSchedFileFailure"/>
184:             <VariableRef name="amSetDebugLevelReceived"/>
185:             <VariableRef name="amSetDebugLevelAccepted"/>
186:             <VariableRef name="amSetDebugLevelSuccess"/>
187:             <VariableRef name="amSetDebugLevelFailure"/>
188:             <VariableRef name="amCurrentDebugLevel"/>

```



```

189:         </DataReplyMsg>
190:     </Request>
191:
192:     <Notification>
193:         <DataMsg name="amSOH" id="013" msgArrival="EVENT">
194:             <Qualifier value="1" name="telemetryLevel"></Qualifier>
195:             <VariableRef name="amActivitiesCurrentlyScheduled"/>
196:             <VariableRef name="amActivitiesExecuted"/>
197:             <VariableRef name="amActivitiesExecutedSuccess"/>
198:             <VariableRef name="amActivitiesExecutedFailed"/>
199:             <VariableRef name="amActivitiesDeleted"/>
200:             <VariableRef name="amSchedActivityReceived"/>
201:             <VariableRef name="amSchedActivityAccepted"/>
202:             <VariableRef name="amSchedActivitySuccess"/>
203:             <VariableRef name="amSchedActivityFailure"/>
204:             <VariableRef name="amAdjustActivityTimeReceived"/>
205:             <VariableRef name="amAdjustActivityTimeSuccess"/>
206:             <VariableRef name="amAdjustActivityTimeFailure"/>
207:             <VariableRef name="amExecuteActivityReceived"/>
208:             <VariableRef name="amExecuteActivityAccepted"/>
209:             <VariableRef name="amExecuteActivitySuccess"/>
210:             <VariableRef name="amExecuteActivityFailure"/>
211:             <VariableRef name="amUpdateActivityExecStatusRcvd"/>
212:             <VariableRef name="amUpdateActivityExecStatusActd"/>
213:             <VariableRef name="amUpdateActivityExecStatusSucc"/>
214:             <VariableRef name="amUpdateActivityExecStatusFail"/>
215:             <VariableRef name="amDeleteActivityReceived"/>
216:             <VariableRef name="amDeleteActivityAccepted"/>
217:             <VariableRef name="amDeleteActivitySuccess"/>
218:             <VariableRef name="amDeleteActivityFailure"/>
219:             <VariableRef name="amDeleteAllActivitiesReceived"/>
220:             <VariableRef name="amDeleteAllActivitiesAccepted"/>
221:             <VariableRef name="amDeleteAllActivitiesSuccess"/>
222:             <VariableRef name="amDeleteAllActivitiesFailure"/>
223:             <VariableRef name="amPrepSchedFileReceived"/>
224:             <VariableRef name="amPrepSchedFileAccepted"/>
225:             <VariableRef name="amPrepSchedFileSuccess"/>
226:             <VariableRef name="amPrepSchedFileFailure"/>
227:             <VariableRef name="amSetDebugLevelReceived"/>
228:             <VariableRef name="amSetDebugLevelAccepted"/>
229:             <VariableRef name="amSetDebugLevelSuccess"/>

```

```

230:         <VariableRef name="amSetDebugLevelFailure"/>
231:         <VariableRef name="amCurrentDebugLevel"/>
232:     </DataMsg>
233: </Notification>
234:
235: </Interface>
236: <!-- Panel interface, encapsulating "panel-level" power housekeeping -->
237: <Interface id="8" name="PanelPowerInterface" description="Panel Power Interface">
238:     <Qualifier name="NumberOfPorts" value="4"/>
239:     <Qualifier name="BreakerTripCurrent" value="30.0" units="Amps"/>
240:     <Variable name="PortReference" kind="PowerPortNumber" format="UINT08"/>
241:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
242:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds"/>
243:     <Variable name="PanelVoltage" kind="Voltage" format="INT16" scaleFactor="0.1"
scaleUnits="Volts"/>
244:     <Variable name="BreakerCurrentArray" length="4" kind="Current" format="INT16"
scaleFactor="1.0" scaleUnits="Amps">
245:         <Qualifier name="Panel" value="Array_4"/>
246:     </Variable>
247:     <Variable name="PortPowerStateArray" length="4" kind="enumeration"
format="UINT08">
248:         <Drange name="PowerStateEnum">
249:             <Option name="Open" value="0"/>
250:             <Option name="Closed" value="1"/>
251:         </Drange>
252:     </Variable>
253:
254: <!-- Message definitions -->
255: <Notification>
256:     <DataMsg id="1" name="PowerStatus" msgArrival="PERIODIC" msgRate="1" >
257:         <Qualifier name="telemetryLevel" value="1"/>
258:         <VariableRef name="Time"/>
259:         <VariableRef name="SubS"/>
260:         <VariableRef name="PanelVoltage"/>
261:         <VariableRef name="BreakerCurrentArray"/>
262:         <VariableRef name="PortPowerStateArray"/>
263:     </DataMsg>
264: </Notification>
265:
266: <!-- Commands and services -->
267: <Command>

```

```

268:         <CommandMsg id="2" name="PortPowerOn" >
269:             <VariableRef name="PortReference"/>
270:         </CommandMsg>
271:     </Command>
272:
273:     <Command>
274:         <CommandMsg id="3" name="PortPowerOff" >
275:             <VariableRef name="PortReference"/>
276:         </CommandMsg>
277:     </Command>
278:
279: </Interface>
280:
281:
282: <Interface id="9" name="RoboHubInterface" description="Interface for a robust hub">
283:     <!-- Properties of the hub and variables in "hub" scope, not "port" scope -->
284:     <Qualifier name="NumberOfPorts" value="8"/>
285:     <Qualifier name="PortTripCurrent" value="4.5" units="Amps"/>
286:     <Variable      name="PortReference"      kind="HubPortNumber"      format="UINT08"
description="Refers to a port for events and commands"/>
287:     <Variable  name="PortEnumeration"  kind="HubEnumerationStatus"  format="UINT08"
description="Used to indicate hub enumeration state"/>
288:     <Variable  name="PPS_Status"  kind="PpsStatus"  format="UINT08"  description="Used to
indicate PPS status"/>
289:     <Variable      name="SetTripCurrentVal"      kind="TripCurrent"      format="INT16"
scaleFactor="0.1" defaultValue="10" scaleUnits="Amps"/>
290:     <Variable name="PortPowerState" kind="PortPowerState" format="UINT08">
291:         <Drange name="PowerStateEnum">
292:             <Option name="Open" value="0"/>
293:             <Option name="Closed" value="1"/>
294:             <Option name="HardTrip" value="2"/>
295:             <Option name="SoftTrip" value="3"/>
296:         </Drange>
297:     </Variable>
298:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
299:     <Variable  kind="SubSeconds"  name="SubS"  units="Counts"  format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
300:
301:     <!-- Variables in "port" scope, declared for each port as arrays of length N_Ports -->
302:     <Variable  name="PortVoltageArray"  length="8"  kind="Voltage"  format="INT16"
scaleFactor="0.1" scaleUnits="Volts">
303:         <Qualifier name="HubPort" value="Array_8"/>

```

```

304:     </Variable>
305:     <Variable name="PortCurrentArray" length="8" kind="Current" format="INT16"
scaleFactor="0.1" scaleUnits="Amps">
306:         <Qualifier name="HubPort" value="Array_8"/>
307:     </Variable>
308:     <Variable name="SoftCurrentLimitArray" length="8" kind="TripCurrent" format="INT16"
scaleFactor="0.1" defaultValue="10" scaleUnits="Amps">
309:         <Qualifier name="HubPort" value="Array_8"/>
310:     </Variable>
311:     <Variable name="PortPowerStateArray" length="8" kind="boolean" format="UINT08">
312:         <Qualifier name="HubPort" value="Array_8"/>
313:         <Drange name="PowerStateEnumArray">
314:             <Option name="Open" value="0"/>
315:             <Option name="Closed" value="1"/>
316:             <Option name="HardTrip" value="2"/>
317:             <Option name="SoftTrip" value="3"/>
318:         </Drange>
319:     </Variable>
320:
321:     <!-- Message Definitions -->
322:     <Notification>
323:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1">
324:             <Qualifier name="telemetryLevel" value="1"/>
325:             <VariableRef name="Time"/>
326:             <VariableRef name="SubS"/>
327:             <VariableRef name="PortPowerStateArray"/>
328:             <VariableRef name="PortVoltageArray"/>
329:             <VariableRef name="PortCurrentArray"/>
330:         </DataMsg>
331:     </Notification>
332:
333:     <Notification>
334:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
335:             <Qualifier name="telemetryLevel" value="1"/>
336:             <VariableRef name="Time"/>
337:             <VariableRef name="SubS"/>
338:             <VariableRef name="PortReference"/>
339:             <VariableRef name="PortPowerState"/>
340:         </DataMsg>
341:     </Notification>
342:

```

```

343:     <!-- Commands and services -->
344:     <Command>
345:         <CommandMsg id="3" name="ConfigureSoftTrip">
346:             <VariableRef name="PortReference"/>
347:             <VariableRef name="SetTripCurrentVal"/>
348:         </CommandMsg>
349:     </Command>
350:
351:     <Command>
352:         <CommandMsg id="4" name="PortPowerOn" >
353:             <VariableRef name="PortReference"/>
354:         </CommandMsg>
355:     </Command>
356:     <Command>
357:         <CommandMsg id="5" name="PortPowerOff" >
358:             <VariableRef name="PortReference"/>
359:         </CommandMsg>
360:     </Command>
361:
362:     <Request>
363:         <CommandMsg id="6" name="GetHubStatus"/>
364:         <DataReplyMsg id="7" name="HubStatusReply">
365:             <VariableRef name="Time"/>
366:             <VariableRef name="SubS"/>
367:             <VariableRef name="PortEnumeration"/>
368:             <VariableRef name="PPS_Status"/>
369:         </DataReplyMsg>
370:     </Request>
371:
372:     <Request>
373:         <CommandMsg id="8" name="GetSoftTripLimits"/>
374:         <DataReplyMsg id="9" name="SoftLimitSettingsReply">
375:             <VariableRef name="Time"/>
376:             <VariableRef name="SubS"/>
377:             <VariableRef name="SoftCurrentLimitArray"/>
378:         </DataReplyMsg>
379:     </Request>
380:
381: </Interface>
382:
383:

```

```

384:    <!-- Component Safety Interface, defining and facilitating monitoring of device health and safety
-->
385:    <Interface name="CmpSafety" id="10">
386:        <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
387:        <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
388:        <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
389:        <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
390:        <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
391:        <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds"/>
392:        <Variable name="DeviceTemperature" kind="temperature" format="INT16"
scaleFactor="1.0" scaleUnits="degC"/>
393:        <Variable name="PanelTemperatureArray" length="8" kind="temperature" format="INT16"
scaleFactor="1.0" scaleUnits="degC">
394:            <Qualifier name="PanelTemperatureChannel" value="Array_8"/>
395:        </Variable>
396:
397:        <Notification>
398:            <DataMsg name="DeviceTemp" id="1" msgArrival="PERIODIC" msgRate="1">
399:                <Qualifier name="telemetryLevel" value="1"/>
400:                <VariableRef name="Time"/>
401:                <VariableRef name="SubS"/>
402:                <VariableRef name="DeviceTemperature"/>
403:                <VariableRef name="PanelTemperatureArray"/>
404:            </DataMsg>
405:        </Notification>
406:
407:    </Interface>
408:    <Interface name="ActivityManagerInterface" id="15">
409:
410:        <Variable name="amActivityId" kind="ID" format="UINT32"></Variable>
411:        <Variable name="amActivityType" kind="tbd" format="UINT08"
length="33"></Variable><!-- 33-byte, null terminated string -->
412:        <Variable name="amActivityPriority" kind="tbd" format="UINT16">
413:            <Drange name="amActivityPriorityEnum">
414:                <Option value="0" name="LOW"/>
415:                <Option value="1" name="MEDIUM"/>
416:                <Option value="2" name="HIGH"/>
417:            </Drange>
418:        </Variable>
419:        <Variable name="amActivityBeginTime" kind="Time" format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies as soon as possible -->

```

```

420:      <Variable      name="amActivityDuration"      kind="Time"      format="FLOAT64"
units="s"></Variable>
421:      <Variable      name="amActivityNotBeforeTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies now -->
422:      <Variable      name="amActivityNotAfterTime"      kind="Time"      format="FLOAT64"
units="s"></Variable><!-- 0.0 signifies no limit -->
423:      <Variable name="amAllowConcurrentActivity" kind="boolean" format="UINT08">
424:          <Drange name="amAllowConcurrentActivityEnum">
425:              <Option value="0" name="NO"/>
426:              <Option value="1" name="YES"/>
427:          </Drange>
428:      </Variable>
429:      <Variable name="amScheduledActivityId" kind="ID" format="UINT32"></Variable>
430:      <Variable name="amStatus" kind="Status" format="INT16">
431:          <Drange name="amStatusEnum">
432:              <Option value="0" name="FAILURE"/>
433:              <Option value="1" name="SUCCESS"/>
434:          </Drange>
435:      </Variable>
436:      <Variable name="amExecStatus" kind="Status" format="INT16">
437:          <Drange name="amExecStatusEnum">
438:              <Option value="0" name="FAILURE"/>
439:              <Option value="1" name="SUCCESS"/>
440:              <Option value="2" name="NOT_EXECUTED"/>
441:          </Drange>
442:      </Variable>
443:      <Variable name="amDebugLevel" kind="tbd" format="UINT16"></Variable>
444:      <Variable name="amCurrentDebugLevel" kind="tbd" format="UINT16"></Variable>
445:      <Variable      name="amActivitiesCurrentlyScheduled"      kind="tbd"
format="UINT16"></Variable>
446:      <Variable name="amActivitiesExecuted" kind="tbd" format="UINT16"></Variable>
447:      <Variable name="amActivitiesExecutedSuccess" kind="tbd" format="UINT16"></Variable>
448:      <Variable name="amActivitiesExecutedFailed" kind="tbd" format="UINT16"></Variable>
449:      <Variable name="amActivitiesDeleted" kind="tbd" format="UINT16"></Variable>
450:      <Variable name="amSchedActivityReceived" kind="tbd" format="UINT16"></Variable>
451:      <Variable name="amSchedActivityAccepted" kind="tbd" format="UINT16"></Variable>
452:      <Variable name="amSchedActivitySuccess" kind="tbd" format="UINT16"></Variable>
453:      <Variable name="amSchedActivityFailure" kind="tbd" format="UINT16"></Variable>
454:      <Variable      name="amAdjustActivityTimeReceived"      kind="tbd"
format="UINT16"></Variable>
455:      <Variable      name="amAdjustActivityTimeAccepted"      kind="tbd"
format="UINT16"></Variable>

```

456: <Variable name="amAdjustActivityTimeSuccess" kind="tbd"
 format="UINT16"></Variable>
 457: <Variable name="amAdjustActivityTimeFailure" kind="tbd"
 format="UINT16"></Variable>
 458: <Variable name="amExecuteActivityReceived" kind="tbd" format="UINT16"></Variable>
 459: <Variable name="amExecuteActivityAccepted" kind="tbd" format="UINT16"></Variable>
 460: <Variable name="amExecuteActivitySuccess" kind="tbd" format="UINT16"></Variable>
 461: <Variable name="amExecuteActivityFailure" kind="tbd" format="UINT16"></Variable>
 462: <Variable name="amUpdateActivityExecStatusRcvd" kind="tbd"
 format="UINT16"></Variable>
 463: <Variable name="amUpdateActivityExecStatusActd" kind="tbd"
 format="UINT16"></Variable>
 464: <Variable name="amUpdateActivityExecStatusSucc" kind="tbd"
 format="UINT16"></Variable>
 465: <Variable name="amUpdateActivityExecStatusFail" kind="tbd"
 format="UINT16"></Variable>
 466: <Variable name="amDeleteActivityReceived" kind="tbd" format="UINT16"></Variable>
 467: <Variable name="amDeleteActivityAccepted" kind="tbd" format="UINT16"></Variable>
 468: <Variable name="amDeleteActivitySuccess" kind="tbd" format="UINT16"></Variable>
 469: <Variable name="amDeleteActivityFailure" kind="tbd" format="UINT16"></Variable>
 470: <Variable name="amDeleteAllActivitiesReceived" kind="tbd"
 format="UINT16"></Variable>
 471: <Variable name="amDeleteAllActivitiesAccepted" kind="tbd"
 format="UINT16"></Variable>
 472: <Variable name="amDeleteAllActivitiesSuccess" kind="tbd"
 format="UINT16"></Variable>
 473: <Variable name="amDeleteAllActivitiesFailure" kind="tbd" format="UINT16"></Variable>
 474: <Variable name="amPrepSchedFileReceived" kind="tbd" format="UINT16"></Variable>
 475: <Variable name="amPrepSchedFileAccepted" kind="tbd" format="UINT16"></Variable>
 476: <Variable name="amPrepSchedFileSuccess" kind="tbd" format="UINT16"></Variable>
 477: <Variable name="amPrepSchedFileFailure" kind="tbd" format="UINT16"></Variable>
 478: <Variable name="amSetDebugLevelReceived" kind="tbd" format="UINT16"></Variable>
 479: <Variable name="amSetDebugLevelAccepted" kind="tbd" format="UINT16"></Variable>
 480: <Variable name="amSetDebugLevelSuccess" kind="tbd" format="UINT16"></Variable>
 481: <Variable name="amSetDebugLevelFailure" kind="tbd" format="UINT16"></Variable>
 482:
 483: <!-- Variables for resources required are tbd -->
 484:
 485: <Request>
 486: <CommandMsg name="amSchedActivity" id="001" description="Schedule a mission or
 housekeeping activity">
 487: <VariableRef name="amActivityId"/>
 488: <VariableRef name="amActivityType"/>


```

489:         <VariableRef name="amActivityPriority"/>
490:         <VariableRef name="amActivityBeginTime"/>
491:         <VariableRef name="amActivityDuration"/>
492:         <VariableRef name="amAllowConcurrentActivity"/>
493:         <VariableRef name="amActivityNotBeforeTime"/>
494:         <VariableRef name="amActivityNotAfterTime"/>
495:
496:         <!-- VariableRefs for resources are tbd -->
497:
498:     </CommandMsg>
499:     <DataReplyMsg name="amSchedActivityReply" id="002">
500:         <VariableRef name="amScheduledActivityId"/>
501:         <VariableRef name="amStatus"/>
502:     </DataReplyMsg>
503: </Request>
504:
505: <Request>
506:     <CommandMsg name="amAdjustActivityTime" id="003" description="Adjust the
scheduled time of a mission or housekeeping activity">
507:         <VariableRef name="amActivityId"/>
508:         <VariableRef name="amActivityBeginTime"/>
509:         <VariableRef name="amActivityDuration"/>
510:     </CommandMsg>
511:     <DataReplyMsg name="amAdjustActivityTimeReply" id="004">
512:         <VariableRef name="amActivityId"/>
513:         <VariableRef name="amStatus"/>
514:     </DataReplyMsg>
515: </Request>
516:
517: <Command>
518:     <CommandMsg name="amExecuteActivity" id="005" description="Execute a scheduled
mission or housekeeping activity">
519:         <VariableRef name="amActivityId"/>
520:     </CommandMsg>
521: </Command>
522:
523: <Command>
524:     <CommandMsg name="amUpdateActivityExecStatus" id="006" description="Update
activity execution statistics">
525:         <VariableRef name="amActivityId"/>
526:         <VariableRef name="amExecStatus"/>
527:     </CommandMsg>

```

```

528:         </Command>
529:
530:         <Command>
531:             <CommandMsg name="amDeleteActivity" id="007" description="Delete a scheduled
activity">
532:                 <VariableRef name="amActivityId"/>
533:             </CommandMsg>
534:         </Command>
535:
536:         <Command>
537:             <CommandMsg name="amDeleteAllActivities" id="008" description="Delete all
scheduled activities"></CommandMsg>
538:         </Command>
539:
540:         <Command>
541:             <CommandMsg name="amPrepSchedFile" id="009" description="Prepare an activities
schedule for downlink"></CommandMsg>
542:         </Command>
543:
544:         <Command>
545:             <CommandMsg name="amSetDebugLevel" id="010" description="Set the debug log
verbosity level">
546:                 <VariableRef name="amDebugLevel"/>
547:             </CommandMsg>
548:         </Command>
549:
550:         <Request>
551:             <CommandMsg name="amGetSoh" id="011"></CommandMsg>
552:             <DataReplyMsg name="amGetSohReply" id="012">
553:                 <VariableRef name="amActivitiesCurrentlyScheduled"/>
554:                 <VariableRef name="amActivitiesExecuted"/>
555:                 <VariableRef name="amActivitiesExecutedSuccess"/>
556:                 <VariableRef name="amActivitiesExecutedFailed"/>
557:                 <VariableRef name="amActivitiesDeleted"/>
558:                 <VariableRef name="amSchedActivityReceived"/>
559:                 <VariableRef name="amSchedActivityAccepted"/>
560:                 <VariableRef name="amSchedActivitySuccess"/>
561:                 <VariableRef name="amSchedActivityFailure"/>
562:                 <VariableRef name="amAdjustActivityTimeReceived"/>
563:                 <VariableRef name="amAdjustActivityTimeSuccess"/>
564:                 <VariableRef name="amAdjustActivityTimeFailure"/>
565:                 <VariableRef name="amExecuteActivityReceived"/>

```

```

566:         <VariableRef name="amExecuteActivityAccepted"/>
567:         <VariableRef name="amExecuteActivitySuccess"/>
568:         <VariableRef name="amExecuteActivityFailure"/>
569:         <VariableRef name="amUpdateActivityExecStatusRcvd"/>
570:         <VariableRef name="amUpdateActivityExecStatusActd"/>
571:         <VariableRef name="amUpdateActivityExecStatusSucc"/>
572:         <VariableRef name="amUpdateActivityExecStatusFail"/>
573:         <VariableRef name="amDeleteActivityReceived"/>
574:         <VariableRef name="amDeleteActivityAccepted"/>
575:         <VariableRef name="amDeleteActivitySuccess"/>
576:         <VariableRef name="amDeleteActivityFailure"/>
577:         <VariableRef name="amDeleteAllActivitiesReceived"/>
578:         <VariableRef name="amDeleteAllActivitiesAccepted"/>
579:         <VariableRef name="amDeleteAllActivitiesSuccess"/>
580:         <VariableRef name="amDeleteAllActivitiesFailure"/>
581:         <VariableRef name="amPrepSchedFileReceived"/>
582:         <VariableRef name="amPrepSchedFileAccepted"/>
583:         <VariableRef name="amPrepSchedFileSuccess"/>
584:         <VariableRef name="amPrepSchedFileFailure"/>
585:         <VariableRef name="amSetDebugLevelReceived"/>
586:         <VariableRef name="amSetDebugLevelAccepted"/>
587:         <VariableRef name="amSetDebugLevelSuccess"/>
588:         <VariableRef name="amSetDebugLevelFailure"/>
589:         <VariableRef name="amCurrentDebugLevel"/>
590:     </DataReplyMsg>
591: </Request>
592:
593: <Notification>
594:     <DataMsg name="amSOH" id="013" msgArrival="EVENT">
595:         <Qualifier value="1" name="telemetryLevel"></Qualifier>
596:         <VariableRef name="amActivitiesCurrentlyScheduled"/>
597:         <VariableRef name="amActivitiesExecuted"/>
598:         <VariableRef name="amActivitiesExecutedSuccess"/>
599:         <VariableRef name="amActivitiesExecutedFailed"/>
600:         <VariableRef name="amActivitiesDeleted"/>
601:         <VariableRef name="amSchedActivityReceived"/>
602:         <VariableRef name="amSchedActivityAccepted"/>
603:         <VariableRef name="amSchedActivitySuccess"/>
604:         <VariableRef name="amSchedActivityFailure"/>
605:         <VariableRef name="amAdjustActivityTimeReceived"/>
606:         <VariableRef name="amAdjustActivityTimeSuccess"/>

```

```

607:         <VariableRef name="amAdjustActivityTimeFailure"/>
608:         <VariableRef name="amExecuteActivityReceived"/>
609:         <VariableRef name="amExecuteActivityAccepted"/>
610:         <VariableRef name="amExecuteActivitySuccess"/>
611:         <VariableRef name="amExecuteActivityFailure"/>
612:         <VariableRef name="amUpdateActivityExecStatusRcvd"/>
613:         <VariableRef name="amUpdateActivityExecStatusActd"/>
614:         <VariableRef name="amUpdateActivityExecStatusSucc"/>
615:         <VariableRef name="amUpdateActivityExecStatusFail"/>
616:         <VariableRef name="amDeleteActivityReceived"/>
617:         <VariableRef name="amDeleteActivityAccepted"/>
618:         <VariableRef name="amDeleteActivitySuccess"/>
619:         <VariableRef name="amDeleteActivityFailure"/>
620:         <VariableRef name="amDeleteAllActivitiesReceived"/>
621:         <VariableRef name="amDeleteAllActivitiesAccepted"/>
622:         <VariableRef name="amDeleteAllActivitiesSuccess"/>
623:         <VariableRef name="amDeleteAllActivitiesFailure"/>
624:         <VariableRef name="amPrepSchedFileReceived"/>
625:         <VariableRef name="amPrepSchedFileAccepted"/>
626:         <VariableRef name="amPrepSchedFileSuccess"/>
627:         <VariableRef name="amPrepSchedFileFailure"/>
628:         <VariableRef name="amSetDebugLevelReceived"/>
629:         <VariableRef name="amSetDebugLevelAccepted"/>
630:         <VariableRef name="amSetDebugLevelSuccess"/>
631:         <VariableRef name="amSetDebugLevelFailure"/>
632:         <VariableRef name="amCurrentDebugLevel"/>
633:     </DataMsg>
634: </Notification>
635:
636: </Interface>
637:
638: </xTEDS>
639:

```

File: sdm/app/test/DMTTests/GenericSearchTest.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <signal.h>
4: #include <unistd.h>
5: #include <ctype.h>
6: #include "../common/message/SDMSearch.h"
7: #include "../common/message/SDMSearchReply.h"
8: #include "../common/MessageManager/MessageManager.h"
9:
10:
11: void SigHandler(int signum);
12: long myPort;
13:
14: SDMSearch Request;
15:
16: bool Menu();
17:
18: int main (int argc, char ** argv)
19: {
20:     MessageManager mm;
21:     char buf[BUFSIZE];
22:     SDMSearchReply ReplyMsg;
23:     unsigned char type;
24:     char InfoSource[40];
25:     char MessageID[40];
26:     SDMInit(argc, argv);
27:     signal(SIGINT,SigHandler);
28:
29:     myPort = getPort();
30:     if (myPort == SDM_PM_NOT_AVAILABLE)
31:     {
32:         myPort = 4059;
33:     }
34:     mm.Async_Init(myPort);
35:
36:     printf("-----SDMSearch Tester----- \n");
37:
38:     bool MenuFinished = false;
39:
```

```

40: while (!MenuFinished)
41:     MenuFinished = Menu();
42:
43: printf("Sending request... \n");
44: Request.destination.setPort(myPort);
45: Request.Send();
46:
47: int count = 1;
48: bool Finished = false;
49: if (Request.reply > SDM_SEARCH_CURRENT)
50: {
51:     printf("Press CTRL+C to quit. \n");
52:     sleep(1);
53: }
54: while (!Finished || Request.reply > SDM_SEARCH_CURRENT)
55: {
56:     type = mm.BlockGetMessage(buf);
57:     switch (type)
58:     {
59:         case SDM_SearchReply:
60:         {
61:             long length = ReplyMsg.Unmarshal(buf);
62:             if (length == SDM_NO_FURTHER_DATA_PROVIDER)
63:                 Finished = true;
64:             else
65:             {
66:                 ReplyMsg.source.IDToString(InfoSource, sizeof(InfoSource));
67:                 printf("(%d) SearchReply received (%s): \n", count++, InfoSource);
68:
69:                 printf("\t---Captured Text Portion:--- \n");
70:
71:                 bool nullFound = false;
72:                 for (unsigned int i = 0; i < sizeof(ReplyMsg.captured_matches); i++)
73:                 {
74:                     if (isprint(ReplyMsg.captured_matches[i]))
75:                     {
76:                         printf("%c", ReplyMsg.captured_matches[i]);
77:                         nullFound = false;
78:                     }
79:                     else if (ReplyMsg.captured_matches[i] == '\0')
80:                     {

```

```

81:             printf("0");
82:             if (nullFound)
83:             {
84:                 printf(" \n");
85:                 break;
86:             }
87:             nullFound = true;
88:         }
89:
90:     }
91: }
92:     break;
93: }
94:     default:
95:         printf("Unexpected message (%d). \n",type);
96:     }
97: }
98: printf("End of SDMSearch messages. \n");
99: printf("-----SDMSearch Tester Finished----- \n");
100:     return 0;
101: }
102:
103: bool Menu()
104: {
105:     unsigned char Item;
106:     printf("(1) Change \"reply \" type \n");
107:     printf("(2) Change \"id \" \n");
108:     printf("(3) Change \"reg_expr \" \n");
109:     printf("(4) Submit SDMSearch \n");
110:     //Get the selection
111:     printf("->");
112:     scanf("%hhu",& Item);
113:
114:     switch(Item)
115:     {
116:         case 1:     //Input for reply
117:         {
118:             int ReplyType;
119:             printf(" \nReply Type: \n");
120:             printf("(1) \tSDM_SEARCH_CURRENT \n");
121:             printf("(2) \tSDM_SEARCH_CURRENT_AND_FUTURE \n");

```

```

122:     printf("(3) \tSDM_SEARCH_CANCEL \n");
123:     printf("->");
124:     scanf("%d",&ReplyType);
125:
126:     switch (ReplyType)
127:     {
128:         case 1:
129:             Request.reply = SDM_SEARCH_CURRENT;
130:             break;
131:         case 2:
132:             Request.reply = SDM_SEARCH_CURRENT_AND_FUTURE;
133:             break;
134:         case 3:
135:             Request.reply = SDM_SEARCH_CANCEL;
136:             break;
137:         default:
138:             printf("Invalid choice, using SDM_SEARCH_CURRENT. \n");
139:             break;
140:     }
141: }
142: break;
143: case 2:    //Input for id
144: {
145:     short Rid;
146:     printf(" \nEnter the \'id \': ");
147:     scanf("%hd",&Rid);
148:     Request.id = Rid;
149: }
150: break;
151: case 3:    //Input for reg_expr
152: {
153:     printf(" \nEnter the \'reg_expr \': ");
154:     scanf("%s", Request.reg_expr);
155: }
156: break;
157:
158: break;
159: case 4:    //Finish the message building
160: {
161:     return true;
162: }

```



```
163:     break;
164:     default:
165:         break;
166: }
167: //User did not finish
168: return false;
169: }
170:
171: void SigHandler(int signum)
172: {
173:
174:     _exit(0);
175: }
```

File: sdm/app/test/DMTTests/SubscriptionTests/Makefile

```
1: include ../../../../Makefile.defs
2:
3: .PHONY: all clean distclean
4:
5: BUILDTARGETS=SubCancelProducer SubCancelTest
6:
7: all: $(BUILDTARGETS)
8:
9: SubCancelTest: SubCancelTest.cpp
10: $(CXX) $(CXXFLAGS) -o $@ $< -L../../../../../common -lSDM -lpthread
11:
12: SubCancelProducer: SubCancelProducer.cpp
13: $(CXX) $(CXXFLAGS) -o $@ $< -L../../../../../common -lSDM -lpthread
14:
15: clean:
16:
17: distclean: clean
18: rm -f $(BUILDTARGETS) *~
```

File: sdm/app/test/DMTests/SubscriptionTests/SubCancelProducer.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMSubreqst.h"
3: #include "../common/message/SDMDeletesub.h"
4: #include "../common/message/SDMCancelxTEDS.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMRegInfo.h"
7: #include "../common/message/SDMData.h"
8: #include "../common/message/SDMConsume.h"
9: #include "../common/message/SDMCancel.h"
10: #include "../common/message/SDMService.h"
11: #include "../common/SubscriptionManager/SubscriptionManager.h"
12: #include "../common/MessageManager/MessageManager.h"
13: #include "../common/message/SDMHeartbeat.h"
14: #include <string.h>
15: #include <sys/types.h>
16: #include <sys/stat.h>
17: #include <fcntl.h>
18: #include <unistd.h>
19: #include <stdio.h>
20: #include <stdlib.h>
21: #include <pthread.h>
22:
23: #include <string>
24:
25: using namespace std;
26:
27: string producerName = "producerone";
28:
29: void RegisterxTEDS(const string& xteds);
30: void CancelxTEDS();
31: void* Publisher(void *);
32: void* Listener(void *);
33: void ReqReg();
34: void ReqRegData();
35: string GetXtedsString(const char* strApplicationName);
36: void RequestSensorId();
37: string GetProviderString();
38: string GetNameString();
39: void CancelNotifications();
```

```

40:
41: SubscriptionManager subscriptions;
42: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER;
43: long my_port;
44:
45: bool bXtedsRegistered = false;
46: bool bProducerOne = false;
47:
48: const unsigned int ID_CHECK_EXISTING_DEVICE = 5;
49: const unsigned int ID_DATA = 6;
50: const unsigned int ID_TM_INFO = 7;
51:
52: SDMComponent_ID dataProvider;
53: SDMComponent_ID myId;
54: SDMMessage_ID dataId;
55: SDMMessage_ID DMPidToSid(1, 8);
56: SDMMessage_ID DMPidToSidReply(1, 9);
57: SDMMessage_ID TMStatusMsg(1, 1);
58:
59: bool bDeviceAlreadyExists = false;
60: bool bMyIdSet = false;
61: const unsigned int THREAD_STACK_SIZE = 128000;
62:
63: int main(int argc, char** argv)
64: {
65:     pthread_t ListenerThread;
66:     pthread_t PublisherThread;
67:
68:     SDMInit(argc, argv);
69:     my_port = getPort();
70:     if(my_port == SDM_PM_NOT_AVAILABLE)
71:     {
72:         printf("No PM is available to get port from! \n");
73:         return 0;
74:     }
75:
76:     pthread_attr_t threadAttr;
77:     pthread_attr_init(&threadAttr);
78:     pthread_attr_setstacksize(&threadAttr, THREAD_STACK_SIZE);
79:
80:     pthread_create(&ListenerThread, &threadAttr, &Listener, NULL);

```

```

81: usleep(100);
82: ReqReg();
83:
84: //usleep(100);
85: pthread_create(&PublisherThread,&threadAttr,&Publisher,NULL);
86: pthread_join(PublisherThread,NULL);
87: CancelxTEDS();
88: CancelNotifications();
89: pthread_cancel(ListenerThread);
90: pthread_join(ListenerThread,NULL);
91: }
92:
93: void* Publisher(void * args)
94: {
95: int published = 0;
96: short data;
97: while(published < 10)
98: {
99:     data = (short)(rand()&0x00FF);
100:     char bufdata[2];
101:     PUT_SHORT(bufdata, data);
102:     pthread_mutex_lock(&subscription_mutex);
103:     if (subscriptions.Publish(1,1,bufdata,2))
104:     {
105:         published++;
106:     }
107:     pthread_mutex_unlock(&subscription_mutex);
108:     printf("Produced %d \tPublished %d / 10 \n",data,published);
109:     sleep(1);
110: }
111: return NULL;
112: }
113:
114: void* Listener(void * args)
115: {
116:     char buf[BUFSIZE];
117:     SDMSubreqst sub;
118:     SDMDeletesub del;
119:     MessageManager mm;
120:     mm.Async_Init(my_port);
121:     while(1)

```

```

122:  {
123:      pthread_testcancel();
124:      if(mm.IsReady())
125:      {
126:          SendHeartbeat();
127: #ifdef WIN32
128:          switch(mm.GetMsg(buf))
129: #else
130:          switch(mm.GetMessage(buf))
131: #endif
132:      {
133:          case SDM_Subreqst:
134:              sub.Unmarshal(buf);
135:              printf("Subscription Rec'd for %d \n",sub.msg_id.getInterfaceMessagePair());
fflush(NULL);
136:              pthread_mutex_lock(&subscription_mutex);
137:              subscriptions.AddSubscription(sub);
138:              pthread_mutex_unlock(&subscription_mutex);
139:              break;
140:          case SDM_Deletesub:
141:              printf("Cancel Rec'd \n");
142:              del.Unmarshal(buf);
143:              pthread_mutex_lock(&subscription_mutex);
144:              subscriptions.RemoveSubscription(del);
145:              pthread_mutex_unlock(&subscription_mutex);
146:              break;
147:          case SDM_RegInfo:
148:              {
149:                  printf ("SDM_RegInfo \n");
150:                  SDMRegInfo msgRegInfo;
151:                  if (msgRegInfo.Unmarshal(buf) == SDM_NO_FURTHER_DATA_PROVIDER)
152:                  {
153:                      if (!bDeviceAlreadyExists && !bXtedsRegistered)
154:                      {
155:                          bProducerOne = true;
156:                          RegisterxTEDS(GetXtedsString("producerone"));
157:                          RequestSensorId();
158:                      }
159:                  }
160:                  else if (!bXtedsRegistered)
161:                  {

```

```

162:         // The other producer is registered, register ourselves with a
163:         // different name
164:         bDeviceAlreadyExists = true;
165:         RegisterxTEDS(GetXtedsString("producertwo"));
166:         RequestSensorId();
167:     }
168:     else if (msgRegInfo.id == ID_DATA)
169:     {
170:         dataProvider = msgRegInfo.source;
171:         dataId = msgRegInfo.msg_id;
172:
173:         SDMConsume msgConsume;
174:         msgConsume.source = msgRegInfo.source;
175:         msgConsume.msg_id = msgRegInfo.msg_id;
176:         msgConsume.destination = myId;
177:         msgConsume.Send();
178:     }
179:     else if (msgRegInfo.id == ID_TM_INFO)
180:     {
181:         printf("Found TM notification, requesting... \n");
182:         SDMConsume msgConsume;
183:
184:         TaskManager = msgRegInfo.source;
185:         msgConsume.destination = myId;
186:         msgConsume.source = TaskManager;
187:         msgConsume.msg_id = TMStatusMsg;
188:         msgConsume.Send();
189:     }
190: }
191: break;
192: case SDM_Data:
193: {
194:     SDMDData msgData;
195:     printf("%s received data \n", GetNameString().c_str());
196:
197:     msgData.Unmarshal(buf);
198:     if (msgData.msg_id == DMPidToSidReply)
199:     {
200:         unsigned long sid = GET_ULONG(msgData.msg + 4);
201:         myId.setSensorID(sid);
202:         myId.setPort(my_port);

```

```

203:         bMyIdSet = true;
204:         printf("Recieved my sensor id from DM (%lu) \n", sid);
205:
206:         ReqRegData();
207:     }
208:     else if (msgData.msg_id == TMStatusMsg)
209:     {
210:         printf("TM status message received. \n");
211:     }
212: }
213:     break;
214: default:
215:     printf("Invalid Message found! \n");
216:     fflush(NULL);
217:     break;
218: }
219: }
220: else
221: {
222:     usleep(100000);
223: }
224: }
225: return NULL;
226: }
227:
228: void RegisterxTEDS(const string& xteds)
229: {
230:     // create an xTEDS registration message
231:     SDMxTEDS msgXteds;
232:
233:     strcpy(msgXteds.xTEDS, xteds.c_str());
234:
235:     // set the id of this application
236:     msgXteds.source.setSensorID(1);
237:     msgXteds.source.setPort(my_port);
238:     printf("Registering %s xTEDS on port %ld \n", GetNameString().c_str(), my_port);
239:     // register with the SDM
240:     msgXteds.Send();
241:
242:     bXtedsRegistered = true;
243: }

```



```

244:
245: void CancelxTEDS()
246: {
247:     SDMCancelxTEDS cancel;
248:     printf("Canceling xTEDS \n");
249:     cancel.source.setSensorID(1);
250:     cancel.source.setPort(my_port);
251:     cancel.Send();
252: }
253:
254: void CancelNotifications()
255: {
256:     SDMCancel msgCancel;
257:
258:     msgCancel.msg_id = TMStatusMsg;
259:     msgCancel.source = TaskManager;
260:     msgCancel.destination = myId;
261:
262:     msgCancel.Send();
263: }
264:
265: void ReqReg()
266: {
267:     printf("ReqReg() \n");
268:     SDMReqReg msgReqReg;
269:
270:     strcpy(msgReqReg.device, producerName.c_str());
271:     msgReqReg.destination.setPort(my_port);
272:     msgReqReg.id = ID_CHECK_EXISTING_DEVICE;
273:
274:     msgReqReg.Send();
275: }
276:
277: string GetProviderString()
278: {
279:     if (bProducerOne)
280:         return "producertwo";
281:     return "producerone";
282: }
283:
284: string GetNameString()

```

```

285: {
286:     if (bProducerOne)
287:         return "producerone";
288:     return "producertwo";
289: }
290:
291: void ReqRegData()
292: {
293:     SDMReqReg msgReqReg;
294:
295:     msgReqReg.destination.setPort(my_port);
296:     strcpy(msgReqReg.device, GetProviderString().c_str());
297:     strcpy(msgReqReg.item_name, "all");
298:     msgReqReg.id = ID_DATA;
299:     msgReqReg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
300:
301:     msgReqReg.Send();
302:
303:     strcpy(msgReqReg.device, "TaskManager");
304:     strcpy(msgReqReg.item_name, "Status");
305:     msgReqReg.id = ID_TM_INFO;
306:     msgReqReg.reply = SDM_REQREG_CURRENT;
307:     msgReqReg.Send();
308: }
309:
310: void RequestSensorId()
311: {
312:     SDMService msgService;
313:
314:     msgService.command_id = DMPidToSid;
315:     msgService.source.setSensorID(1);
316:     msgService.destination.setPort(my_port);
317:     PUT_ULONG(msgService.data, PID);
318:     msgService.length = sizeof(PID);
319:
320:     msgService.Send();
321: }
322:
323: string GetXtedsString(const char* strApplicationName)
324: {
325:     string appName = strApplicationName;

```

```

326: string producerXteds = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n \
327: <xTEDS version= \"2.0 \" name= \"Producer_xTEDS \"> \
328:   \n \t<Application name= \"\" + appName + \" \" kind= \"data \"/> \
329:   \n \t<Interface name= \"Producer_Interface \" id= \"1 \"> \
330:     \n \t<Variable name= \"data \" format= \"UINT16 \" \
331:     kind= \"data \"/> \
332:     \n \t<Notification> \
333:       \n \t \t<DataMsg name= \"all \" id= \"1 \" \
334:       msgArrival= \"PERIODIC \" msgRate= \"1 \"> \
335:       \n \t \t \t<VariableRef name= \"data \"/> \
336:       \n \t \t</DataMsg> \
337:       \n \t</Notification> \
338:     \n \t</Interface> \
339:   \n</xTEDS> \n";
340:
341: return producerXteds;
342: }
343:

```

File: sdm/app/test/DMTTests/SubscriptionTests/SubCancelTest.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5:
6: #include "../common/MessageManager/MessageManager.h"
7: #include "../common/message/SDMReqReg.h"
8: #include "../common/message/SDMRegInfo.h"
9: #include "../common/message/SDMCommand.h"
10:
11: long lMyPort;
12:
13: void FindTMServices();
14: void DeleteTasks();
15: void CopyTasks();
16: void RegInfoHandler(const SDMRegInfo&);
17:
18: const int ID_START_TASK = 1;
19:
20: SDMMMessage_ID idStartTask;
21: SDMComponent_ID idTaskManager;
22:
23: bool bFinish = false;
24:
25: int main (int argc, char** argv)
26: {
27:     SDMInit(argc, argv);
28:
29:     lMyPort = getPort();
30:     if (lMyPort == SDM_PM_NOT_AVAILABLE)
31:         lMyPort = 5432;
32:
33:     MessageManager mm;
34:     mm.Async_Init(lMyPort);
35:
36:     CopyTasks();
37:     FindTMServices();
38:
39:     char buf[BUFSIZE];
```

```

40: SDMRegInfo msgRegInfo;
41: while (!bFinish)
42: {
43:     if (mm.IsReady())
44:     {
45:         switch(mm.GetMessage(buf))
46:         {
47:             case SDM_RegInfo:
48:             {
49:                 if (msgRegInfo.Unmarshal(buf) != SDM_NO_FURTHER_DATA_PROVIDER)
50:                 {
51:                     RegInfoHandler(msgRegInfo);
52:                 }
53:                 break;
54:             }
55:         }
56:     }
57:     else
58:         sleep(1);
59: }
60:
61: return 0;
62: }
63:
64: void RegInfoHandler(const SDMRegInfo& msgInfo)
65: {
66: if (msgInfo.id == ID_START_TASK)
67: {
68:     idStartTask = msgInfo.msg_id;
69:     idTaskManager = msgInfo.source;
70:
71:     SDMCommand msgCommand;
72:     msgCommand.command_id = idStartTask;
73:     msgCommand.source = idTaskManager;
74:     strcpy(msgCommand.data + 2, "SubCancelProducer");
75:     msgCommand.length = 19;
76:
77:     printf("Starting SubCancelProducer \n");
78:     msgCommand.Send();
79:     sleep(2);
80:

```

```

81:    printf("Starting another SubCancelProducer \n");
82:    msgCommand.Send();
83:
84:    bFinish = true;
85: }
86:
87:
88: }
89:
90: void FindTMServices()
91: {
92:     SDMReqReg msgRequest;
93:
94:     strcpy(msgRequest.device, "TaskManager");
95:     strcpy(msgRequest.item_name, "StartTask");
96:     msgRequest.id = ID_START_TASK;
97:     msgRequest.destination.setPort(IMyPort);
98:     msgRequest.Send();
99: }
100:
101: void CopyTasks()
102: {
103:     system("cp SubCancelProducer ../../../../tm/");
104: }
105:
106: void DeleteTasks()
107: {
108:     system("rm -f ../../../../tm/SubCancelProducer");
109: }
110:

```

File: sdm/app/test/DMTests/xTEDSRegTests/PowerController.h

```
1: #ifndef _POWERCONTROLLER_XTEDS_H
2: #define _POWERCONTROLLER_XTEDS_H
3:
4: #define _STRING_POWERCONTROLLER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"Battery_XTEDS \"\" \
8: "version= \"2.0\">\" \
9: "<Application name= \"PowerController \" kind= \"PowerControl \" description= \"Routine Control of
Spacecraft Power \">\" \
10: "<Qualifier name= \"Author \" value= \"Design_Net \"/>\" \
11: "<Qualifier name= \"Version \" value= \"1.1 \"/>\" \
12: "<Qualifier name= \"RevisionDate \" value= \"03-27-2007 \"/>\" \
13: "</Application>\" \
14: "" \
15: "<Interface name= \"PowerStatusInterface \" id= \"1\">\" \
16: "<Variable name= \"Time \" kind= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
17: "<Variable name= \"SubS \" kind= \"SubSeconds \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \" />\" \
18: "<Variable name= \"BusVoltage \" kind= \"BusVoltage \" format= \"FLOAT32 \" units= \"Volts \"
description= \"Spacecraft Bus Voltage \" />\" \
19: "<Variable name= \"EnergyStoreCapacity \" kind= \"Capacity \" format= \"FLOAT32 \" units=
\"Watt-Hrs \" description= \"Energy Storage Capacity\">\" \
20: "<Qualifier name= \"Scope \" value= \"System \"/>\" \
21: "<Qualifier name= \"Category \" value= \"Max \"/>\" \
22: "</Variable>\" \
23: "<Variable name= \"EnergyStorePowerCapacity \" kind= \"Capacity \" format= \"FLOAT32 \" units=
\"Watts \" description= \"Energy Storage Power Capacity\">\" \
24: "<Qualifier name= \"Scope \" value= \"System \"/>\" \
25: "<Qualifier name= \"Category \" value= \"Max \"/>\" \
26: "</Variable>\" \
27: "<Variable name= \"CollectionPowerCapacity \" kind= \"Capacity \" format= \"FLOAT32 \" units=
\"Watts \" description= \"Energy Collection Power Capacity\">\" \
28: "<Qualifier name= \"Scope \" value= \"System \"/>\" \
29: "<Qualifier name= \"Category \" value= \"Max \"/>\" \
30: "</Variable>\" \
31: "<Variable name= \"DissipationPowerCapacity \" kind= \"Capacity \" format= \"FLOAT32 \" units=
\"Watts \" description= \"Energy Dissipation Power Capacity\">\" \
32: "<Qualifier name= \"Scope \" value= \"System \"/>\" \
```

33: "<Qualifier name= \"Category \" value= \"Max \"/>\" \

34: "</Variable>\" \

35: "<Variable name= \"StoredEnergy \" kind= \"Quantity \" format= \"FLOAT32 \" units= \"Watt-Hrs \" description= \"Energy Store Stored Energy \">\" \

36: "<Qualifier name= \"Scope \" value= \"System \"/>\" \

37: "<Qualifier name= \"Category \" value= \"Instantaneous \"/>\" \

38: "</Variable>\" \

39: "<Variable name= \"EnergyStoreStateOfCharge \" kind= \"Percentage \" format= \"FLOAT32 \" units= \"Percent \" description= \"Energy Store State Of Charge \">\" \

40: "<Qualifier name= \"Scope \" value= \"System \"/>\" \

41: "<Qualifier name= \"Category \" value= \"Instantaneous \"/>\" \

42: "</Variable>\" \

43: "<Variable name= \"EnergyStorePower \" kind= \"Power \" format= \"FLOAT32 \" units= \"Watts \" description= \"Energy Store Power \">\" \

44: "<Qualifier name= \"Scope \" value= \"System \"/>\" \

45: "<Qualifier name= \"Category \" value= \"Instantaneous \"/>\" \

46: "</Variable>\" \

47: "<Variable name= \"CollectionPower \" kind= \"Power \" format= \"FLOAT32 \" units= \"Watts \" description= \"Energy Collection Power \">\" \

48: "<Qualifier name= \"Scope \" value= \"System \"/>\" \

49: "<Qualifier name= \"Category \" value= \"Instantaneous \"/>\" \

50: "</Variable>\" \

51: "<Variable name= \"DissipationPower \" kind= \"Power \" format= \"FLOAT32 \" units= \"Watts \" description= \"Energy Dissipation Power \">\" \

52: "<Qualifier name= \"Scope \" value= \"System \"/>\" \

53: "<Qualifier name= \"Category \" value= \"Instantaneous \"/>\" \

54: "</Variable>\" \

55: "<Variable name= \"DevicePower \" kind= \"Power \" format= \"FLOAT32 \" units= \"Watts \" description= \"Device Power \">\" \

56: "<Qualifier name= \"Scope \" value= \"System \"/>\" \

57: "<Qualifier name= \"Category \" value= \"Instantaneous \"/>\" \

58: "</Variable>\" \

59: "<Variable name= \"TimeToYellowLimit \" kind= \"Duration \" format= \"UINT32 \" units= \"Seconds \" description= \"Time until Yellow Limit Condition is Reached \">\" \

60: "<Qualifier name= \"Scope \" value= \"PowerController \"/>\" \

61: "<Qualifier name= \"Target \" value= \"PowerAdvisoryCondition \"/>\" \

62: "</Variable>\" \

63: "<Variable name= \"TimeToRedLimit \" kind= \"Duration \" format= \"UINT32 \" units= \"Seconds \" description= \"Time until Red Limit Condition is Reached \">\" \

64: "<Qualifier name= \"Scope \" value= \"PowerManagement \"/>\" \

65: "<Qualifier name= \"Target \" value= \"PowerWarningCondition \"/>\" \

66: "</Variable>\" \

67: "<Variable name= \"SensorId \" kind= \"Identification \" format= \"UINT32 \" description= \"Unique identifier of spacecraft component \"/>\" \"

68: "<Variable name= \"HubNumber \" kind= \"Identification \" format= \"UINT32 \" description= \"Hub number \"/>\" \"

69: "<Variable name= \"PortNumber \" kind= \"Identification \" format= \"UINT32 \" description= \"Port number \"/>\" \"

70: "" \"

71: "<!-- Notifications -->\" \"

72: "<Notification>\" \"

73: "<DataMsg name= \"PowerState \" description= \"System-Level Power State \" id= \"1 \" msgArrival= \"PERIODIC \"/>\" \"

74: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \"

75: "<VariableRef name= \"Time \" />\" \"

76: "<VariableRef name= \"SubS \" />\" \"

77: "<VariableRef name= \"BusVoltage \" />\" \"

78: "<VariableRef name= \"EnergyStoreCapacity \" />\" \"

79: "<VariableRef name= \"EnergyStorePowerCapacity \" />\" \"

80: "<VariableRef name= \"CollectionPowerCapacity \" />\" \"

81: "<VariableRef name= \"DissipationPowerCapacity \" />\" \"

82: "<VariableRef name= \"StoredEnergy \" />\" \"

83: "<VariableRef name= \"EnergyStoreStateOfCharge \" />\" \"

84: "<VariableRef name= \"EnergyStorePower \" />\" \"

85: "<VariableRef name= \"CollectionPower \" />\" \"

86: "<VariableRef name= \"DissipationPower \" />\" \"

87: "<VariableRef name= \"DevicePower \" />\" \"

88: "</DataMsg>\" \"

89: "</Notification>\" \"

90: "<Notification>\" \"

91: "<DataMsg name= \"YellowLimitState \" description= \"Time remaining until yellow limit event \" id= \"2 \" msgArrival= \"PERIODIC \"/>\" \"

92: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \"

93: "<VariableRef name= \"Time \" />\" \"

94: "<VariableRef name= \"SubS \" />\" \"

95: "<VariableRef name= \"TimeToYellowLimit \"/>\" \"

96: "</DataMsg>\" \"

97: "</Notification>\" \"

98: "<Notification>\" \"

99: "<DataMsg name= \"RedLimitState \" description= \"Time remaining until red limit event \" id= \"3 \" msgArrival= \"PERIODIC \"/>\" \"

100: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \"

101: "<VariableRef name= \"Time \" />\" \"

102: "<VariableRef name= \"SubS \" />\" \"

```

103: "<VariableRef name= \"TimeToRedLimit \"/>" \
104: "</DataMsg>" \
105: "</Notification>" \
106: "<Notification>" \
107: "<DataMsg name= \"YellowLimitEvent \" description= \"Yellow limit condition notification \" id= \"4 \" msgArrival= \"EVENT \">" \
108: "<VariableRef name= \"Time \"/>" \
109: "<VariableRef name= \"SubS \"/>" \
110: "<VariableRef name= \"EnergyStoreStateOfCharge \"/>" \
111: "</DataMsg>" \
112: "</Notification>" \
113: "<Notification>" \
114: "<DataMsg name= \"RedLimitEvent \" description= \"Red limit condition notification \" id= \"5 \" msgArrival= \"EVENT \">" \
115: "<VariableRef name= \"Time \"/>" \
116: "<VariableRef name= \"SubS \"/>" \
117: "<VariableRef name= \"EnergyStoreStateOfCharge \"/>" \
118: "</DataMsg>" \
119: "</Notification>" \
120: "<Notification>" \
121: "<DataMsg name= \"PortSoftTripEvent \" description= \"Power hub port soft current trip notification \" id= \"6 \" msgArrival= \"EVENT \">" \
122: "<VariableRef name= \"Time \"/>" \
123: "<VariableRef name= \"SubS \"/>" \
124: "<VariableRef name= \"SensorId \"/>" \
125: "<VariableRef name= \"HubNumber \"/>" \
126: "<VariableRef name= \"PortNumber \"/>" \
127: "</DataMsg>" \
128: "</Notification>" \
129: "<Notification>" \
130: "<DataMsg name= \"PortHardTripEvent \" description= \"Power hub port hard current trip notification \" id= \"7 \" msgArrival= \"EVENT \">" \
131: "<VariableRef name= \"Time \"/>" \
132: "<VariableRef name= \"SubS \"/>" \
133: "<VariableRef name= \"SensorId \"/>" \
134: "<VariableRef name= \"HubNumber \"/>" \
135: "<VariableRef name= \"PortNumber \"/>" \
136: "</DataMsg>" \
137: "</Notification>" \
138: "" \
139: "<!-- Commands -->" \
140: "<Command>" \

```

```

141: "<CommandMsg name= \"ResetSoftTrippedPort \" id= \"8 \">" \
142: "<VariableRef name= \"HubNumber \" />" \
143: "<VariableRef name= \"PortNumber \" />" \
144: "</CommandMsg>" \
145: "</Command>" \
146: "<Command>" \
147: "<CommandMsg name= \"ResetHardTrippedPort \" id= \"9 \">" \
148: "<VariableRef name= \"HubNumber \" />" \
149: "<VariableRef name= \"PortNumber \" />" \
150: "</CommandMsg>" \
151: "</Command>" \
152: "</Interface>" \
153: "" \
154: "<Interface name= \"PowerPlanningInterface \" id= \"2 \">" \
155: "<Variable name= \"PlanId \" kind= \"Identification \" format= \"UINT32 \" description= \"Unique
identifier of planning request \"/>" \
156: "<Variable name= \"Timeout \" kind= \"Duration \" format= \"UINT32 \" description= \"Time until
plan expires if not acknowledged \"/>" \
157: "<Variable name= \"NewTimeToYellowLimit \" kind= \"Duration \" format= \"UINT32 \" units=
\"Seconds \" description= \"Time until Yellow Limit Condition is Reached \"/>" \
158: "<Variable name= \"NewTimeToRedLimit \" kind= \"Duration \" format= \"UINT32 \" units=
\"Seconds \" description= \"Time until Red Limit Condition is Reached \"/>" \
159: "" \
160: "<!-- Requests -->" \
161: "<Request>" \
162: "<CommandMsg name= \"RequestPower \" id= \"1 \">" \
163: "" \
164: "</CommandMsg>" \
165: "<DataReplyMsg name= \"RequestPowerReply \" id= \"2 \">" \
166: "<VariableRef name= \"PlanId \"/>" \
167: "<VariableRef name= \"NewTimeToYellowLimit \"/>" \
168: "<VariableRef name= \"NewTimeToRedLimit \"/>" \
169: "</DataReplyMsg>" \
170: "</Request>" \
171: "" \
172: "<!-- Commands -->" \
173: "<Command>" \
174: "<CommandMsg name= \"AcknowledgePlan \" id= \"3 \">" \
175: "<VariableRef name= \"PlanId \"/>" \
176: "</CommandMsg>" \
177: "</Command>" \
178: "<Command>" \

```

```
179: "<CommandMsg name= \"CancelPlan \" id= \"4 \">" \
180: "<VariableRef name= \"PlanId \"/>" \
181: "</CommandMsg>" \
182: "</Command>" \
183: "</Interface>" \
184: "" \
185: "</xTEDS>" \
186: ""
187:
188: #endif
```

File: sdm/app/test/DMTests/xTEDSRegTests/ADCSController.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                      xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="ADCS_Control_App_xTeds" version="2.1">
4: <Application name="ADCSController" version="1.0" kind="SHAP" description="Implements the top-
level interface to the ADCS" />
5: <Interface name="ADCSInterface" id="1">
6:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
7:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32" scaleFactor=".0001"
scaleUnits="Seconds" />
8:   <Variable name="Rate" kind="attitudeRate" units="rad_s" format="FLOAT64" length="3"
description=" Desired Angular rates about each primary axis of the vehicle" >
9:     <Qualifier name="representation" value="vector" />
10:    <Qualifier name="frameMeasured" value="SVF" />
11:    <Qualifier name="frameResolved" value="SVF" />
12:  </Variable>
13:  <Variable name="TargetObject" kind="target" format="UINT08" description="Target at which the
system should point" >
14:    <Drange name="TargetsEnum">
15:      <Option name="Nadir" value="0"/>
16:      <Option name="Zenith" value="1"/>
17:      <Option name="Sun" value="2"/>
18:      <Option name="Other" value="3"/>
19:    </Drange>
20:  </Variable>
21:  <Variable name="TargetPosition" kind="position" length="3" units="km" format="FLOAT64"
description="ECI position vector of a target at which we wish to point">
22:    <Qualifier name="representation" value="vector"/>
23:    <Qualifier name="frameMeasured" value="ECIMOD"/>
24:    <Qualifier name="frameResolved" value="ECIMOD"/>
25:  </Variable>
26:  <Variable name="TargetVelocity" kind="velocity" length="3" units="m_s" format="FLOAT64"
description="ECI velocity vector of a target at which we wish to point">
27:    <Qualifier name="representation" value="vector"/>
28:    <Qualifier name="frameMeasured" value="ECIMOD"/>
29:    <Qualifier name="frameResolved" value="ECIMOD"/>
30:  </Variable>
31:  <Variable name="TargetAcceleration" kind="acceleration" length="3" units="m_s_s"
format="FLOAT64" description="ECI acceleration vector of a target at which we wish to point">
32:    <Qualifier name="representation" value="vector"/>
```

```

33: <Qualifier name="frameMeasured" value="ECIMOD"/>
34: <Qualifier name="frameResolved" value="ECIMOD"/>
35: </Variable>
36: <Variable name="BoresightObject" kind="boresight" format="UINT08" description="Boresight we
wish to point" >
37: <Drange name="BoresightsEnum">
38: <Option name="SolarPanel" value="0"/>
39: <Option name="PrimaryPayload" value="1"/>
40: <Option name="Other" value="2"/>
41: </Drange>
42: </Variable>
43: <Variable name="BoresightLOS" kind="LOS" length="3" format="FLOAT64" description="LOS
of the Boresight we wish to point">
44: <Qualifier name="representation" value="vector"/>
45: <Qualifier name="frameMeasured" value="SVF"/>
46: <Qualifier name="frameResolved" value="SVF"/>
47: </Variable>
48: <Variable name="Offset" kind="rollPitchYaw" length="3" units="rad" format="FLOAT32"
description="pushbroom offsets from LVLH [rad]"/>
49: <!--Commands for the ADCS System-->
50: <Command>
51: <CommandMsg id="1" name="SetADCSModeStandby" description="Set the ADCS to standby
mode" />
52: </Command>
53: <Command>
54: <CommandMsg id="2" name="SetADCSModeRate" description="Set the ADCS to rate-only
mode">
55: <VariableRef name="Rate"/>
56: </CommandMsg>
57: </Command>
58: <Command>
59: <CommandMsg id="3" name="SetADCSModeTracking" description="Set the ADCS to tracking
mode">
60: <VariableRef name="BoresightObject"/>
61: <VariableRef name="TargetObject"/>
62: <VariableRef name="Offset"/>
63: <!-- Boresight LOS only used if BoresightObject is "other", send 0's otherwise-->
64: <VariableRef name="BoresightLOS"/>
65: <!--Target Specifications only used if TargetObject is "other", send 0's otherwise-->
66: <VariableRef name="TargetPosition"/>
67: <VariableRef name="TargetVelocity"/>
68: <VariableRef name="TargetAcceleration"/>

```

```
69:  </CommandMsg>
70:  </Command>
71:  <Command>
72:    <CommandMsg id="4" name="SetADCSModeMomentumDump" description="Set the ADCS to
momentum dumping mode" />
73:  </Command>
74: </Interface>
75: </xTEDS>
```

File: sdm/app/test/DMTests/xTEDSRegTests/xTEDSRegTest.cpp

```
1: #include "ActivityAgent.h"
2: #include "ActivityManager.h"
3: #include "AdcoleDigitalSS.h"
4: #include "ADCSController.h"
5: #include "AnglAccelToTorque.h"
6: #include "Battery.h"
7: #include "BIT.h"
8: #include "ChargeBatteries.h"
9: #include "CSSAssy.h"
10: #include "DigitalSS.h"
11: #include "DownlinkController.h"
12: #include "GPS_Full.h"
13: #include "GPS.h"
14: #include "IDS.h"
15: #include "iMESA.h"
16: #include "IRU.h"
17: #include "MagTorqueRod.h"
18: #include "OOCE.h"
19: #include "PowerController.h"
20: #include "RoboHub.h"
21: #include "RoboHub_lean.h"
22: #include "RWheelAssy.h"
23: #include "RWheelSingle.h"
24: #include "SolarArray.h"
25: #include "StarCamera.h"
26: #include "Targeting.h"
27: #include "TelemetryHandler.h"
28: #include "ThreeAxisMagnetometer.h"
29: #include "Thruster.h"
30: #include "VehicleService.h"
31:
32: #include "../common/message/SDMxTEDS.h"
33: #include <stdio.h>
34: #include <string.h>
35: #include <unistd.h>
36:
37: const int SLEEP_VAL = 50000;
38:
39: int main (int argc, char** argv)
```



```

40: {
41:  SDMInit(argc, argv);
42:
43:  SDMxTEDS xTEDSMsg;
44:
45:  strcpy(xTEDSMsg.xTEDS, _STRING_ACTIVITYAGENT_XTEDS);
46:  xTEDSMsg.Send();
47:  usleep(SLEEP_VAL);
48:
49:  strcpy(xTEDSMsg.xTEDS, _STRING_ACTIVITYMANAGER_XTEDS);
50:  xTEDSMsg.Send();
51:  usleep(SLEEP_VAL);
52:
53:  strcpy(xTEDSMsg.xTEDS, _STRING_ADCOLE_NRL_DIGITAL_SUN_SENSOR_XTEDS);
54:  xTEDSMsg.Send();
55:  usleep(SLEEP_VAL);
56:
57:  strcpy(xTEDSMsg.xTEDS, _STRING_ADCSCONTROLLER_XTEDS);
58:  xTEDSMsg.Send();
59:  usleep(SLEEP_VAL);
60:
61:  strcpy(xTEDSMsg.xTEDS, _STRING_TORQUECONTROL_XTEDS);
62:  xTEDSMsg.Send();
63:  usleep(SLEEP_VAL);
64:
65:  strcpy(xTEDSMsg.xTEDS, _STRING_BATTERY_XTEDS);
66:  xTEDSMsg.Send();
67:  usleep(SLEEP_VAL);
68:
69:  strcpy(xTEDSMsg.xTEDS, _STRING_EXAMPLE_XTEDS);
70:  xTEDSMsg.Send();
71:  usleep(SLEEP_VAL);
72:
73:  strcpy(xTEDSMsg.xTEDS, _STRING_CHARGEBATTERIES_XTEDS);
74:  xTEDSMsg.Send();
75:  usleep(SLEEP_VAL);
76:
77:  strcpy(xTEDSMsg.xTEDS, _STRING_COARSE_SUN_SENSOR_ASSEMBLY_XTEDS);
78:  xTEDSMsg.Send();
79:  usleep(SLEEP_VAL);
80:

```

```

81: strcpy(xTEDSMsg.xTEDS, _STRING_ADCOLENRLDIGITALSUNSENSOR_XTEDS);
82: xTEDSMsg.Send();
83: usleep(SLEEP_VAL);
84:
85: strcpy(xTEDSMsg.xTEDS, _STRING_DOWNLINKCONTROLLER_XTEDS);
86: xTEDSMsg.Send();
87: usleep(SLEEP_VAL);
88:
89: strcpy(xTEDSMsg.xTEDS, _STRING_FULLGPSRECEIVER_XTEDS);
90: xTEDSMsg.Send();
91: usleep(SLEEP_VAL);
92:
93: strcpy(xTEDSMsg.xTEDS, _STRING_BASICGPSRECEIVER_XTEDS);
94: xTEDSMsg.Send();
95: usleep(SLEEP_VAL);
96:
97: strcpy(xTEDSMsg.xTEDS, _STRING_INTELLIGENTDATASTORE_XTEDS);
98: xTEDSMsg.Send();
99: usleep(SLEEP_VAL);
100:
101:  strcpy(xTEDSMsg.xTEDS, _STRING_IMESA_XTEDS);
102:  xTEDSMsg.Send();
103:  usleep(SLEEP_VAL);
104:
105:  strcpy(xTEDSMsg.xTEDS, _STRING_LN200S_IRU_XTEDS);
106:  xTEDSMsg.Send();
107:  usleep(SLEEP_VAL);
108:
109:  strcpy(xTEDSMsg.xTEDS, _STRING_BASICMAGNETICTORQUEROD_XTEDS);
110:  xTEDSMsg.Send();
111:  usleep(SLEEP_VAL);
112:
113:  strcpy(xTEDSMsg.xTEDS, _STRING_OOCE_XTEDS);
114:  xTEDSMsg.Send();
115:  usleep(SLEEP_VAL);
116:
117:  strcpy(xTEDSMsg.xTEDS, _STRING_POWERCONTROLLER_XTEDS);
118:  xTEDSMsg.Send();
119:  usleep(SLEEP_VAL);
120:
121:  strcpy(xTEDSMsg.xTEDS, _STRING_ROBOHUB_8_PORT_XTEDS);

```

```

122:  xTEDSMsg.Send();
123:  usleep(SLEEP_VAL);
124:
125:  strcpy(xTEDSMsg.xTEDS, _STRING_REACTIONWHEELASSEMBLY_XTEDS);
126:  xTEDSMsg.Send();
127:  usleep(SLEEP_VAL);
128:
129:  strcpy(xTEDSMsg.xTEDS, _STRING_SINGLE_REACTION_WHEEL_XTEDS);
130:  xTEDSMsg.Send();
131:  usleep(SLEEP_VAL);
132:
133:  strcpy(xTEDSMsg.xTEDS, _STRING_SOLARARRAY_XTEDS);
134:  xTEDSMsg.Send();
135:  usleep(SLEEP_VAL);
136:
137:  strcpy(xTEDSMsg.xTEDS, _STRING_TERMAHE5ASSTARTRACKER_XTEDS);
138:  xTEDSMsg.Send();
139:  usleep(SLEEP_VAL);
140:
141:  strcpy(xTEDSMsg.xTEDS, _STRING_TARGETINGCONTROL_XTEDS);
142:  xTEDSMsg.Send();
143:  usleep(SLEEP_VAL);
144:
145:  strcpy(xTEDSMsg.xTEDS, _STRING_TELEMETRYHANDLER_XTEDS);
146:  xTEDSMsg.Send();
147:  usleep(SLEEP_VAL);
148:
149:  strcpy(xTEDSMsg.xTEDS, _STRING_THREE_AXIS_MAGNETOMETER_XTEDS);
150:  xTEDSMsg.Send();
151:  usleep(SLEEP_VAL);
152:
153:  //strcpy(xTEDSMsg.xTEDS, _STRING_MONOPROPELLANTTHRUSTER_XTEDS);
154:  //xTEDSMsg.Send();
155:  //usleep(SLEEP_VAL);
156:
157:  strcpy(xTEDSMsg.xTEDS, _STRING_VEHICLEINFORMATIONSERVICE_XTEDS);
158:  xTEDSMsg.Send();
159:  usleep(SLEEP_VAL);
160:
161:  return 0;
162: }

```

File: sdm/app/test/DMTests/xTEDSRegTests/CSSAssy.h

```
1: #ifndef _COARSE_SUN_SENSOR_ASSEMBLY_XTEDS_H
2: #define _COARSE_SUN_SENSOR_ASSEMBLY_XTEDS_H
3:
4: #define _STRING_COARSE_SUN_SENSOR_ASSEMBLY_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"DNet_CoarseSSAssy_xTEDS \"\" \
8:   \"version= \"2.1 \">\" \
9:   \"<Device name= \"Coarse_sun_sensor_assembly \" kind= \"cssa \" description= \"An assembly of 4
coarse sun sensors \" />\" \
10:   \"\" \
11:   \"<Interface id= \"1 \" name= \"CSSAssemblyInterface \" description= \"Aggregate-level interface for
the assembly \" >\" \
12:   \"<Qualifier name= \"headID \" value= \"0 \" />\" \
13:   \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
14:   \"<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \" />\" \
15:   \"<Variable name= \"SolutionAvailable \" kind= \"boolean \" format= \"UINT08 \">\" \
16:   \"<Drange name= \"SolutionEnum \" >\" \
17:   \"<Option name= \"Yes \" value= \"1 \" />\" \
18:   \"<Option name= \"No \" value= \"0 \" />\" \
19:   \"</Drange>\" \
20:   \"</Variable>\" \
21:   \"<Variable name= \"dataQuality \" kind= \"DataQuality \" format= \"UINT08 \">\" \
22:   \"<Drange name= \"DataQualityEnum \">\" \
23:   \"<Option name= \"dataBad \" value= \"0 \" description= \"data is garbage or NaN. \" />\" \
24:   \"<Option name= \"dataPoor \" value= \"1 \" description= \"data quality is poor. \" />\" \
25:   \"<Option name= \"dataGood \" value= \"2 \" description= \"data is good. \" />\" \
26:   \"</Drange>\" \
27:   \"</Variable>\" \
28:   \"<Variable name= \"AngleToSun \" kind= \"sunLOS \" format= \"FLOAT32 \" length= \"2 \" />\" \
29:   \"<Notification>\" \
30:   \"<DataMsg id= \"1 \" name= \"SunSolution \" msgArrival= \"PERIODIC \" msgRate= \"10 \">\" \
31:   \"<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
32:   \"<VariableRef name= \"Time \" />\" \
33:   \"<VariableRef name= \"SubS \" />\" \
34:   \"<VariableRef name= \"SolutionAvailable \" />\" \
35:   \"<VariableRef name= \"dataQuality \" />\" \
```

```

36: "<VariableRef name= \"AngleToSun \"/>" \
37: "</DataMsg>" \
38: "</Notification>" \
39: "</Interface>" \
40: "" \
41: "<Interface id= \"2 \" name= \"CSSAssyHeadInterface \" description= \"Messaging for a single CSS
head \">" \
42: "<Qualifier name= \"headID \" value= \"1 \"/>" \
43: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
44: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
45: "<Variable kind= \"ID \" name= \"HeadID \" format= \"UINT08 \"/>" \
46: "<Variable kind= \"boresightLOS \" name= \"HeadLOS \" format= \"FLOAT32 \" length= \"3 \"/>" \
47: "<Variable kind= \"sunCOS \" name= \"SunCos \" format= \"FLOAT32 \"/>" \
48: "<Variable name= \"SunPresence \" kind= \"Valid \" format= \"UINT08 \">" \
49: "<Drange name= \"Sun_PresenceEnum \">" \
50: "<Option name= \"SunPresent \" value= \"1 \" description= \"This sensor can see the sun \"/>" \
51: "<Option name= \"SunNotPresent \" value= \"0 \" description= \"Sun not visible from this sensor \"
/>" \
52: "</Drange>" \
53: "</Variable>" \
54: "<Request>" \
55: "<CommandMsg id= \"1 \" name= \"getHeadLOS \" description= \"Returns the orientation of this
head in the sensor frame \"/>" \
56: "<DataReplyMsg id= \"2 \" name= \"HeadLOSReply \">" \
57: "<VariableRef name= \"HeadID \"/>" \
58: "<VariableRef name= \"HeadLOS \"/>" \
59: "</DataReplyMsg>" \
60: "</Request>" \
61: "<Notification>" \
62: "<DataMsg id= \"3 \" name= \"sunConeAngle \" msgArrival= \"PERIODIC \" msgRate= \"10 \"
description= \"Observation of the sun from this sensor head \">" \
63: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
64: "<VariableRef name= \"Time \"/>" \
65: "<VariableRef name= \"SubS \"/>" \
66: "<VariableRef name= \"SunPresence \"/>" \
67: "<VariableRef name= \"SunCos \"/>" \
68: "</DataMsg>" \
69: "</Notification>" \
70: "</Interface>" \
71: "" \

```

```

72: "<Interface id= \"3 \" name= \"CSSAssyHeadInterface \" description= \"Messaging for a single CSS
head \" >\" \
73: "<Qualifier name= \"headID \" value= \"2 \"/>\" \
74: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>\" \
75: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>\" \
76: "<Variable kind= \"ID \" name= \"HeadID \" format= \"UINT08 \"/>\" \
77: "<Variable kind= \"boresightLOS \" name= \"HeadLOS \" format= \"FLOAT32 \" length= \"3 \"/>\" \
78: "<Variable kind= \"sunCOS \" name= \"SunCos \" format= \"FLOAT32 \"/>\" \
79: "<Variable name= \"SunPresence \" kind= \"Valid \" format= \"UINT08 \">\" \
80: "<Drange name= \"Sun_PresenceEnum \">\" \
81: "<Option name= \"SunPresent \" value= \"1 \" description= \"This sensor can see the sun \" />\" \
82: "<Option name= \"SunNotPresent \" value= \"0 \" description= \"Sun not visible from this sensor \"
/>\" \
83: "</Drange>\" \
84: "</Variable>\" \
85: "<Request>\" \
86: "<CommandMsg id= \"1 \" name= \"getHeadLOS \" description= \"Returns the orientation of this
head in the sensor frame \" />\" \
87: "<DataReplyMsg id= \"2 \" name= \"HeadLOSReply \">\" \
88: "<VariableRef name= \"HeadID \"/>\" \
89: "<VariableRef name= \"HeadLOS \"/>\" \
90: "</DataReplyMsg>\" \
91: "</Request>\" \
92: "<Notification>\" \
93: "<DataMsg id= \"3 \" name= \"sunConeAngle \" msgArrival= \"PERIODIC \" msgRate= \"10 \"
description= \"Observation of the sun from this sensor head \">\" \
94: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
95: "<VariableRef name= \"Time \"/>\" \
96: "<VariableRef name= \"SubS \"/>\" \
97: "<VariableRef name= \"SunPresence \"/>\" \
98: "<VariableRef name= \"SunCos \"/>\" \
99: "</DataMsg>\" \
100: "</Notification>\" \
101: "</Interface>\" \
102: "" \
103: "<Interface id= \"4 \" name= \"CSSAssyHeadInterface \" description= \"Messaging for a single CSS
head \" >\" \
104: "<Qualifier name= \"headID \" value= \"3 \"/>\" \
105: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>\" \
106: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>\" \

```

```

107: "<Variable kind= \"ID \" name= \"HeadID \" format= \"UINT08 \" />\" \
108: "<Variable kind= \"boresightLOS \" name= \"HeadLOS \" format= \"FLOAT32 \" length= \"3 \" />\" \
109: "<Variable kind= \"sunCOS \" name= \"SunCos \" format= \"FLOAT32 \" />\" \
110: "<Variable name= \"SunPresence \" kind= \"Valid \" format= \"UINT08 \">\" \
111: "<Drange name= \"Sun_PresenceEnum \">\" \
112: "<Option name= \"SunPresent \" value= \"1 \" description= \"This sensor can see the sun \" />\" \
113: "<Option name= \"SunNotPresent \" value= \"0 \" description= \"Sun not visible from this sensor \" />\" \
114: "</Drange>\" \
115: "</Variable>\" \
116: "<Request>\" \
117: "<CommandMsg id= \"1 \" name= \"getHeadLOS \" description= \"Returns the orientation of this head in the sensor frame \" />\" \
118: "<DataReplyMsg id= \"2 \" name= \"HeadLOSReply \">\" \
119: "<VariableRef name= \"HeadID \"/>\" \
120: "<VariableRef name= \"HeadLOS \"/>\" \
121: "</DataReplyMsg>\" \
122: "</Request>\" \
123: "<Notification>\" \
124: "<DataMsg id= \"3 \" name= \"sunConeAngle \" msgArrival= \"PERIODIC \" msgRate= \"10 \" description= \"Observation of the sun from this sensor head \">\" \
125: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
126: "<VariableRef name= \"Time \"/>\" \
127: "<VariableRef name= \"SubS \"/>\" \
128: "<VariableRef name= \"SunPresence \"/>\" \
129: "<VariableRef name= \"SunCos \"/>\" \
130: "</DataMsg>\" \
131: "</Notification>\" \
132: "</Interface>\" \
133: "" \
134: "<Interface id= \"5 \" name= \"CSSAssyHeadInterface \" description= \"Messaging for a single CSS head \">\" \
135: "<Qualifier name= \"headID \" value= \"4 \"/>\" \
136: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
137: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
138: "<Variable kind= \"ID \" name= \"HeadID \" format= \"UINT08 \" />\" \
139: "<Variable kind= \"boresightLOS \" name= \"HeadLOS \" format= \"FLOAT32 \" length= \"3 \" />\" \
140: "<Variable kind= \"sunCOS \" name= \"SunCos \" format= \"FLOAT32 \" />\" \
141: "<Variable name= \"SunPresence \" kind= \"Valid \" format= \"UINT08 \">\" \

```

```

142: "<Drange name= \"Sun_PresenceEnum \"/>" \
143: "<Option name= \"SunPresent \" value= \"1 \" description= \"This sensor can see the sun \"/>" \
144: "<Option name= \"SunNotPresent \" value= \"0 \" description= \"Sun not visible from this sensor \"/>" \
145: "</Drange>" \
146: "</Variable>" \
147: "<Request>" \
148: "<CommandMsg id= \"1 \" name= \"getHeadLOS \" description= \"Returns the orientation of this head in the sensor frame \"/>" \
149: "<DataReplyMsg id= \"2 \" name= \"HeadLOSReply \"/>" \
150: "<VariableRef name= \"HeadID \"/>" \
151: "<VariableRef name= \"HeadLOS \"/>" \
152: "</DataReplyMsg>" \
153: "</Request>" \
154: "<Notification>" \
155: "<DataMsg id= \"3 \" name= \"sunConeAngle \" msgArrival= \"PERIODIC \" msgRate= \"10 \" description= \"Observation of the sun from this sensor head \"/>" \
156: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
157: "<VariableRef name= \"Time \"/>" \
158: "<VariableRef name= \"SubS \"/>" \
159: "<VariableRef name= \"SunPresence \"/>" \
160: "<VariableRef name= \"SunCos \"/>" \
161: "</DataMsg>" \
162: "</Notification>" \
163: "</Interface>" \
164: "" \
165: "<Interface name= \"DevPwr \" id= \"6 \"/>" \
166: "<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \"/>" \
167: "<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \"/>" \
168: "<Qualifier name= \"CurrentHiWarning \" value= \"0.2 \" units= \"A \"/>" \
169: "<Qualifier name= \"CurrentHiKeepout \" value= \"0.3 \" units= \"A \"/>" \
170: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
171: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
172: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \">" \
173: "<Drange name= \"DevPwrStateEnum \"/>" \
174: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \"/>" \
175: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \"/>" \
176: "</Drange>" \
177: "</Variable>" \
178: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \">" \

```


179: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to DevPwrStateEnumeration. \"/>\" \
 180: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />\" \
 181: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />\" \
 182: "</Drange>\" \
 183: "</Variable>\" \
 184: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length= \"2 \" />\" \
 185: "<Command>\" \
 186: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \" />\" \
 187: "<VariableRef name= \"DevPwrStateSet \" />\" \
 188: "</CommandMsg>\" \
 189: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \" />\" \
 190: "<VariableRef name= \"Time \" />\" \
 191: "<VariableRef name= \"SubS \" />\" \
 192: "<VariableRef name= \"DevPwrState \" />\" \
 193: "<VariableRef name= \"DevPwrStateSet \" />\" \
 194: "</FaultMsg>\" \
 195: "</Command>\" \
 196: "<Notification>\" \
 197: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \" />\" \
 198: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
 199: "<VariableRef name= \"Time \" />\" \
 200: "<VariableRef name= \"SubS \" />\" \
 201: "<VariableRef name= \"DevPwrState \" />\" \
 202: "<VariableRef name= \"DevPwrStateSet \" />\" \
 203: "</DataMsg>\" \
 204: "</Notification>\" \
 205: "<Request>\" \
 206: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />\" \
 207: "<DataReplyMsg name= \"powerInMode \" id= \"5 \" />\" \
 208: "<VariableRef name= \"modePowers \" />\" \
 209: "</DataReplyMsg>\" \
 210: "</Request>\" \
 211: "</Interface>\" \
 212: "" \
 213: "<Interface name= \"CmpSafety \" id= \"7 \" />\" \
 214: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \" />\" \
 215: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \" />\" \
 216: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \" />\" \
 217: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \" />\" \

```

218: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
219: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \" />\" \
220: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units=
\"degC \" />\" \
221: "<Request>\" \
222: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />\" \
223: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">\" \
224: "<VariableRef name= \"Time \" />\" \
225: "<VariableRef name= \"SubS \" />\" \
226: "<VariableRef name= \"DeviceTemperature \" />\" \
227: "</DataReplyMsg>\" \
228: "</Request>\" \
229: "<Notification>\" \
230: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">\" \
231: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
232: "<VariableRef name= \"Time \" />\" \
233: "<VariableRef name= \"SubS \" />\" \
234: "<VariableRef name= \"DeviceTemperature \" />\" \
235: "</DataMsg>\" \
236: "</Notification>\" \
237: "</Interface>\" \
238: "" \
239: "</xTEDS>\" \
240: ""
241:
242: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/AnglAccelToTorque.h

```
1: #ifndef _TORQUECONTROL_XTEDS_H
2: #define _TORQUECONTROL_XTEDS_H
3:
4: #define _STRING_TORQUECONTROL_XTEDS \
5: "<?xml version= \"1.0 \" encoding= \"utf-8 \" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation=  \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
   \"Torque_Control_xTeds \"\" \
8:   "version= \"2.0 \">\" \
9:   "<Application name= \"TorqueControl \" version= \"1.0 \" kind= \"AHAP \" description= \"Takes
   commands for angular acceleration, converts to torque, and distributes torque commands to actuators \"
   />\" \
10:   "" \
11:   "<Interface name= \"TorqueControlInterface \" id= \"1 \">\" \
12:   "<Variable name= \"AnglAccel \" kind= \"angularAcceleration \" length= \"3 \" format= \"FLOAT64
   \" description= \"Commanded angular acceleration \">\" \
13:   "<Qualifier name= \"representation \" value= \"vector \"/>\" \
14:   "<Qualifier name= \"frameMeasured \" value= \"SVF \"/>\" \
15:   "<Qualifier name= \"frameResolved \" value= \"SVF \"/>\" \
16:   "</Variable>\" \
17:   "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
18:   "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
   scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
19:   "<Variable kind= \"Torque \" name= \"X \" units= \"nm \" format= \"FLOAT32 \" description= \"X
   component of torque to apply \" />\" \
20:   "<Variable kind= \"Torque \" name= \"Y \" units= \"nm \" format= \"FLOAT32 \" description= \"Y
   component of torque to apply \" />\" \
21:   "<Variable kind= \"Torque \" name= \"Z \" units= \"nm \" format= \"FLOAT32 \" description= \"Z
   component of torque to apply \" />\" \
22:   "<Command>\" \
23:   "<CommandMsg id= \"1 \" name= \"SCAngularAccelCmd \" description= \"Command to apply
   angular acceleration to the spacecraft \">\" \
24:   "<VariableRef name= \"AnglAccel \"/>\" \
25:   "</CommandMsg>\" \
26:   "</Command>\" \
27:   "<Notification>\" \
28:   "<DataMsg id= \"2 \" name= \"SVTorqueToApply \" description= \"Torque equivalent to requested
   angular acceleration \" msgArrival= \"PERIODIC \" msgRate= \"10 \">\" \
29:   "<VariableRef name= \"Time \"/>\" \
30:   "<VariableRef name= \"SubS \"/>\" \
```

```
31: "<VariableRef name= \"X \"/>" \
32: "<VariableRef name= \"Y \"/>" \
33: "<VariableRef name= \"Z \"/>" \
34: "</DataMsg>" \
35: "</Notification>" \
36: "</Interface>" \
37: "</xTEDS>" \
38: ""
39:
40: #endif
```

File: sdm/app/test/DMTests/xTEDSRegTests/SolarArray.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="SolarArray_XTEDS"
4:   version="2.0">
5:   <Device name="SolarArray" kind="sa" description="a device that provides electrical power" >
6:     <Qualifier name="Manufacturer" value="MsiFitsArray"/>
7:     <Qualifier name="Model" value="1.2.3"/>
8:     <Qualifier name="SerialNumber" value="90210"/>
9:   </Device>
10:
11:   <Interface name="SolarArrayInterface" id="1">
12:     <Qualifier name="CellType" value="GaAs"/>
13:     <Qualifier name="BOLPower" value="80" units="W"/>
14:     <Qualifier name="PerformanceDegradation" value="5" units="percent_year"/>
15:     <Qualifier name="MaxPowerVoltage" value="20" units="Volts"/>
16:     <Qualifier name="MaxPowerCurrent" value="10" units="Amps"/>
17:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
18:     <Variable   kind="SubSeconds"   name="SubS"   units="Counts"   format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
19:     <Variable   name="SATemp"   kind="Temperature"   format="FLOAT32"   units="degC"
description="ambient temperature of array" />
20:     <Variable   name="SACurrent"   kind="Current"   format="FLOAT32"   units="Amps"
description="charging is positive; discharging is negative" />
21:     <Variable   name="SADesiredCurrent"   kind="SetCurrent"   format="FLOAT32"   units="Amps"
description="charging is positive; discharging is negative" />
22:     <Variable   name="SAUnregVoltage"   kind="Voltage"   format="FLOAT32"   units="Volts"
description="voltage at battery terminals" />
23:     <Variable name="SAOperationState" kind="mode" format="UINT08">
24:       <Drange name="OpStateEnum">
25:         <Option name="Offline" value="0"/>
26:         <Option name="Online" value="1"/>
27:       </Drange>
28:     </Variable>
29:
30:   <Command>
31:     <CommandMsg id="1" name="SetOpState">
32:       <VariableRef name="SAOperationState"/>
33:     </CommandMsg>
```

```

34:     </Command>
35:
36:     <Command>
37:         <CommandMsg id="2" name="SetCurrentLimitOut">
38:             <VariableRef name="SADesiredCurrent"/>
39:         </CommandMsg>
40:     </Command>
41:
42:     <Notification>
43:         <DataMsg      name="ArraySOH"      description="state    of    health"    id="3"
msgArrival="PERIODIC">
44:             <Qualifier name="telemetryLevel" value="1"/>
45:             <Qualifier name="Type" value="SOH" />
46:             <VariableRef name="Time" />
47:             <VariableRef name="SubS" />
48:             <VariableRef name="SATemp" />
49:             <VariableRef name="SACurrent" />
50:             <VariableRef name="SADesiredCurrent"/>
51:             <VariableRef name="SAUnregVoltage" />
52:             <VariableRef name="SAOperationState" />
53:         </DataMsg>
54:     </Notification>
55:
56: </Interface>
57:
58: <Interface id="2" name="DeployerInterface">
59:     <!--description="Interface for a deployment mechanism - in this case the release for the array."-->
60:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
61:     <Variable    kind="SubSeconds"    name="SubS"    units="Counts"    format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
62:     <Variable kind="mode" name="ArmedState" format="UINT08">
63:         <Drange name="ArmedStateEnum">
64:             <Option name="Safe" value="0"/>
65:             <Option name="Armed" value="1"/>
66:         </Drange>
67:     </Variable>
68:     <Variable kind="mode" name="DeployedState" format="UINT08">
69:         <Drange name="DeployedStateEnum">
70:             <Option name="Stowed" value="0"/>
71:             <Option name="Deploying" value="1"/>
72:             <Option name="Deployed" value="2"/>

```

```

73:     </Drange>
74: </Variable>
75: <Command>
76:     <CommandMsg id="1" name="ArmOrDisarmRelease" description="Toggles the safe/armed
state of the deployment mechanism">
77:         <VariableRef name="ArmedState"/>
78:     </CommandMsg>
79: </Command>
80: <Command>
81:     <CommandMsg id="2" name="DeployGo" description="Deploy the array"/>
82: </Command>
83: <Request>
84:     <CommandMsg id="3" name="GetDeployState" />
85:     <DataReplyMsg id="4" name="DeployState">
86:         <VariableRef name="Time"/>
87:         <VariableRef name="SubS"/>
88:         <VariableRef name="ArmedState"/>
89:         <VariableRef name="DeployedState"/>
90:     </DataReplyMsg>
91: </Request>
92: </Interface>
93:
94: <Interface name="CmpSafety" id="3">
95:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
96:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
97:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
98:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
99:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
100:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
101:     <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32"
units="degC" />
102:     <Request>
103:         <CommandMsg name="GetDeviceTemperature" id="1" />
104:         <DataReplyMsg name="DeviceTempReply" id="2">
105:             <VariableRef name="Time" />
106:             <VariableRef name="SubS" />
107:             <VariableRef name="DeviceTemperature"/>
108:         </DataReplyMsg>
109:     </Request>
110:     <Notification>
111:         <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">

```

```
112:         <Qualifier name="telemetryLevel" value="1"/>
113:         <VariableRef name="Time" />
114:         <VariableRef name="SubS" />
115:         <VariableRef name="DeviceTemperature"/>
116:     </DataMsg>
117: </Notification>
118: </Interface>
119: </xTEDS>
```


File: sdm/app/test/DMTests/xTEDSRegTests/Battery.h

```
1: #ifndef _BATTERY_XTEDS_H
2: #define _BATTERY_XTEDS_H
3:
4: #define _STRING_BATTERY_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"Battery_XTEDS \"\" \
8:   \"version= \"2.0 \">\" \
9:   \"<Device name= \"Battery \" kind= \"bat \" description= \"power storage device \">\" \
10:   \"<Qualifier name= \"Manufacturer \" value= \"DNetConceptBattery \"/>\" \
11:   \"<Qualifier name= \"Model \" value= \"1.2.3 \"/>\" \
12:   \"<Qualifier name= \"SerialNumber \" value= \"90210 \"/>\" \
13:   \"</Device>\" \
14:   \"\" \
15:   \"<Interface name= \"BatteryBasicInterface \" id= \"1 \">\" \
16:   \"<Qualifier name= \"Chemistry \" value= \"LiIon \"/>\" \
17:   \"<Qualifier name= \"Capacity \" value= \"30.0 \" units= \"Amp-Hours \"/>\" \
18:   \"<Qualifier name= \"DepthOfDischargeLoWarn \" value= \"20.0 \" units= \"percent \"/>\" \
19:   \"<Qualifier name= \"DepthOfDischargeLoKeepout \" value= \"10.0 \" units= \"percent \"/>\" \
20:   \"<Qualifier name= \"CycleLimit \" value= \"500 \" units= \"cycles \"/>\" \
21:   \"<Variable name= \"BatSOC \" kind= \"dischargeFraction \" format= \"FLOAT32 \" units= \"percent
\" description= \"percent of capacity discharged \" />\" \
22:   \"<Variable name= \"BatTemp \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \"
description= \"ambient temperature inside battery \" />\" \
23:   \"<Variable name= \"BatCycleCount \" kind= \"count \" format= \"INT16 \" units= \"counts \"
description= \"count of discharge/recharge cycles the device has experienced \" />\" \
24:   \"<Variable name= \"BatCurrent \" kind= \"electricalCurrent \" format= \"FLOAT32 \" units= \"A \"
description= \"charging is positive; discharging is negative \" />\" \
25:   \"<Variable name= \"BatUnregVoltage \" kind= \"voltage \" format= \"FLOAT32 \" units= \"V \"
description= \"voltage at battery terminals \" />\" \
26:   \"<Variable name= \"CurrentLimitIn \" kind= \"electricalCurrent \" format= \"FLOAT32 \" units= \"A
\" description= \"Limit on current flowing into battery \" />\" \
27:   \"<Variable name= \"CurrentLimitOut \" kind= \"electricalCurrent \" format= \"FLOAT32 \" units=
\"A \" description= \"Limit on current flowing out of battery \" />\" \
28:   \"<Variable name= \"BatteryState \" kind= \"mode \" format= \"UINT08 \">\" \
29:   \"<Drange name= \"BatteryStateEnum \">\" \
30:   \"<Option name= \"Offline \" value= \"0 \"/>\" \
31:   \"<Option name= \"Online \" value= \"1 \"/>\" \
32:   \"<Option name= \"Calibrate \" value= \"2 \"/>\" \
```

```

33: "</Drange>" \
34: "</Variable>" \
35: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
36: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
37: "" \
38: "<Notification>" \
39: "<DataMsg name= \"BatSOH \" description= \"state of health \" id= \"1 \" msgArrival= \"PERIODIC \">" \
40: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
41: "<VariableRef name= \"Time \" />" \
42: "<VariableRef name= \"SubS \" />" \
43: "<VariableRef name= \"BatteryState \" />" \
44: "<VariableRef name= \"BatSOC \" />" \
45: "<VariableRef name= \"BatTemp \" />" \
46: "<VariableRef name= \"BatCycleCount \" />" \
47: "<VariableRef name= \"BatCurrent \" />" \
48: "<VariableRef name= \"BatUnregVoltage \" />" \
49: "</DataMsg>" \
50: "</Notification>" \
51: "" \
52: "<Command>" \
53: "<CommandMsg id= \"2 \" name= \"SetCurrentLimitIn \">" \
54: "<VariableRef name= \"CurrentLimitIn \" />" \
55: "</CommandMsg>" \
56: "</Command>" \
57: "" \
58: "<Command>" \
59: "<CommandMsg id= \"3 \" name= \"SetCurrentLimitOut \">" \
60: "<VariableRef name= \"CurrentLimitOut \" />" \
61: "</CommandMsg>" \
62: "</Command>" \
63: "" \
64: "<Command>" \
65: "<CommandMsg id= \"4 \" name= \"SetBatteryState \">" \
66: "<VariableRef name= \"BatteryState \" />" \
67: "</CommandMsg>" \
68: "</Command>" \
69: "</Interface>" \
70: "" \
71: "<Interface name= \"CmpSafety \" id= \"3 \">" \

```

```

72: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>" \
73: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>" \
74: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>" \
75: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>" \
76: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
77: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
78: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units=
\"degC \"/>" \
79: "<Request>" \
80: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />" \
81: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">" \
82: "<VariableRef name= \"Time \" />" \
83: "<VariableRef name= \"SubS \" />" \
84: "<VariableRef name= \"DeviceTemperature \"/>" \
85: "</DataReplyMsg>" \
86: "</Request>" \
87: "<Notification>" \
88: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
89: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
90: "<VariableRef name= \"Time \" />" \
91: "<VariableRef name= \"SubS \" />" \
92: "<VariableRef name= \"DeviceTemperature \"/>" \
93: "</DataMsg>" \
94: "</Notification>" \
95: "</Interface>" \
96: "" \
97: "</xTEDS>" \
98: ""
99:
100: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/ActivityManager.h

```
1: #ifndef _ACTIVITYMANAGER_XTEDS_H
2: #define _ACTIVITYMANAGER_XTEDS_H
3:
4: #define _STRING_ACTIVITYMANAGER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"ActivityManagerXTEDS \" description= \"ActivityManager xTEDS \" version= \"3.1 \">\" \
8:   \"\" \
9:   \"<Application name= \"ActivityManager \" kind= \"AutonomyFlightSoftware \" description=
\"Autonomous Tasking Executive (ATE), ActivityManager \"/>\" \
10:   \"\" \
11:   \"<Interface name= \"ActivityManagerInterface \" id= \"1 \" description= \"Basic interface for
scheduling activities and updating their status \">\" \
12:   \"\" \
13:   \"<Variable name= \"ActivityId \" kind= \"ID \" format= \"UINT32 \"/>\" \
14:   \"<Variable name= \"ActivityName \" kind= \"String \" format= \"INT08 \" length= \"33 \"/><!-- 33-
byte, null terminated string -->\" \
15:   \"<Variable name= \"ActivityPriority \" kind= \"tbd \" format= \"UINT08 \" rangeMin= \"0 \"
rangeMax= \"255 \"/><!-- 0 is the lowest priority and 255 is the highest -->\" \
16:   \"<Variable name= \"ActivityBeginTime \" kind= \"Time \" format= \"FLOAT64 \" units= \"s \"/>\" \
17:   \"<Variable name= \"ActivityDuration \" kind= \"Time \" format= \"FLOAT64 \" units= \"s \"/>\" \
18:   \"<Variable name= \"ActivityNotBeforeTime \" kind= \"Time \" format= \"FLOAT64 \" units= \"s
\"/><!-- Earliest begin time. 0.0 signifies now -->\" \
19:   \"<Variable name= \"ActivityNotAfterTime \" kind= \"Time \" format= \"FLOAT64 \" units= \"s
\"/><!-- Latest begin time. 0.0 signifies no limit -->\" \
20:   \"<Variable name= \"AllowConcurrentActivity \" kind= \"boolean \" format= \"UINT08 \">\" \
21:   \"<Drange name= \"AllowConcurrentActivityEnum \">\" \
22:   \"<Option value= \"0 \" name= \"NO \"/>\" \
23:   \"<Option value= \"1 \" name= \"YES \"/>\" \
24:   \"</Drange>\" \
25:   \"</Variable>\" \
26:   \"<Variable name= \"ActivityStatus \" kind= \"Status \" format= \"INT16 \">\" \
27:   \"<Drange name= \"ActivityStatusEnum \">\" \
28:   \"<Option value= \"0 \" name= \"SCHEDULE_FAILURE \"/><!-- The activity could not be scheduled
as requested -->\" \
29:   \"<Option value= \"1 \" name= \"WAITING \"/><!-- The activity has been inserted into the schedule --
>\" \
30:   \"<Option value= \"2 \" name= \"ENABLED \"/><!-- The activity has been sent a command to execute
-->\" \
```

```

31: "<Option value= \"3 \" name= \"TERMINATED \"/><!-- The activity has been sent a command to
abort -->" \
32: "<Option value= \"4 \" name= \"EXECUTING \"/><!-- The activity has indicated that it is currently
executing -->" \
33: "<Option value= \"5 \" name= \"DONE_FAILURE \"/><!-- The activity has indicated that it
completed abnormally -->" \
34: "<Option value= \"6 \" name= \"DONE_SUCCESS \"/><!-- The activity has indicated that it
completed normally -->" \
35: "<Option value= \"7 \" name= \"DONE_NOT_EXECUTED \"/><!-- The activity has indicated that it
terminated without executing -->" \
36: "</Drange>" \
37: "</Variable>" \
38: "<Variable name= \"ExecutionStatus \" kind= \"Status \" format= \"INT16 \" defaultValue= \"4 \">>" \
39: "<Drange name= \"ExecutionStatusEnum \">>" \
40: "<Option value= \"4 \" name= \"EXECUTING \"/><!-- The activity is currently executing -->" \
41: "<Option value= \"5 \" name= \"DONE_FAILURE \"/><!-- The activity has completed abnormally --
>" \
42: "<Option value= \"6 \" name= \"DONE_SUCCESS \"/><!-- The activity has completed normally -->"
\
43: "<Option value= \"7 \" name= \"DONE_NOT_EXECUTED \"/><!-- The activity has terminated
without executing -->" \
44: "</Drange>" \
45: "</Variable>" \
46: "" \
47: "<!-- Variables for resources required are tbd -->" \
48: "" \
49: "<Request>" \
50: "<CommandMsg name= \"ScheduleActivity \" id= \"001 \" description= \"Schedule a mission or
housekeeping activity \">>" \
51: "<VariableRef name= \"ActivityId \"/>" \
52: "<VariableRef name= \"ActivityName \"/>" \
53: "<VariableRef name= \"ActivityPriority \"/>" \
54: "<VariableRef name= \"ActivityDuration \"/>" \
55: "<VariableRef name= \"AllowConcurrentActivity \"/>" \
56: "<VariableRef name= \"ActivityNotBeforeTime \"/>" \
57: "<VariableRef name= \"ActivityNotAfterTime \"/>" \
58: "" \
59: "<!-- VariableRefs for resources are tbd -->" \
60: "" \
61: "</CommandMsg>" \
62: "<DataReplyMsg name= \"ScheduleActivityReply \" id= \"002 \">>" \
63: "<VariableRef name= \"ActivityId \"/>" \
64: "<VariableRef name= \"ActivityStatus \"/>" \

```

65: "</DataReplyMsg>" \
 66: "</Request>" \
 67: "" \
 68: "<Request>" \
 69: "<CommandMsg name= \"UpdateActivity \" id= \"003 \" description= \"Adjust the properties of a scheduled mission or housekeeping activity \">" \
 70: "<VariableRef name= \"ActivityId \"/>" \
 71: "<VariableRef name= \"ActivityPriority \"/>" \
 72: "<VariableRef name= \"ActivityDuration \"/>" \
 73: "<VariableRef name= \"AllowConcurrentActivity \"/>" \
 74: "<VariableRef name= \"ActivityNotBeforeTime \"/>" \
 75: "<VariableRef name= \"ActivityNotAfterTime \"/>" \
 76: "" \
 77: "<!-- VariableRefs for resources are tbd -->" \
 78: "" \
 79: "</CommandMsg>" \
 80: "<DataReplyMsg name= \"UpdateActivityReply \" id= \"004 \">" \
 81: "<VariableRef name= \"ActivityId \"/>" \
 82: "<VariableRef name= \"ActivityStatus \"/>" \
 83: "</DataReplyMsg>" \
 84: "</Request>" \
 85: "" \
 86: "<Command>" \
 87: "<CommandMsg name= \"UpdateActivityStatus \" id= \"005 \" description= \"Update activity execution status \">" \
 88: "<VariableRef name= \"ActivityId \"/>" \
 89: "<VariableRef name= \"ExecutionStatus \"/>" \
 90: "</CommandMsg>" \
 91: "</Command>" \
 92: "" \
 93: "<Command>" \
 94: "<CommandMsg name= \"DeleteActivity \" id= \"006 \" description= \"Delete a scheduled activity \">" \
 95: "<VariableRef name= \"ActivityId \"/>" \
 96: "</CommandMsg>" \
 97: "</Command>" \
 98: "" \
 99: "<Command>" \
 100: "<CommandMsg name= \"SendActivityStatusMsg \" id= \"007 \" description= \"Send the ActivityStatusMsg DataMsg \">" \
 101: "<VariableRef name= \"ActivityId \"/>" \
 102: "</CommandMsg>" \

```

103: "</Command>" \
104: "" \
105: "<Notification>" \
106: "<DataMsg name= \"ActivityStatusMsg\" id= \"008\" msgArrival= \"EVENT \">\" \
107: "<Qualifier value= \"1\" name= \"telemetryLevel \"/>\" \
108: "<VariableRef name= \"ActivityId \"/>\" \
109: "<VariableRef name= \"ActivityName \"/>\" \
110: "<VariableRef name= \"ActivityPriority \"/>\" \
111: "<VariableRef name= \"ActivityBeginTime \"/>\" \
112: "<VariableRef name= \"ActivityDuration \"/>\" \
113: "<VariableRef name= \"AllowConcurrentActivity \"/>\" \
114: "<VariableRef name= \"ActivityNotBeforeTime \"/>\" \
115: "<VariableRef name= \"ActivityNotAfterTime \"/>\" \
116: "<VariableRef name= \"ActivityStatus \"/>\" \
117: "</DataMsg>" \
118: "</Notification>" \
119: "" \
120: "</Interface>" \
121: "" \
122: "<Interface name= \"ActivityManagerScheduleInterface\" id= \"2\" description= \"Additional
messages for schedule maintenance\" >\" \
123: "" \
124: "<Command>" \
125: "<CommandMsg name= \"DeleteAllActivities\" id= \"001\" description= \"Delete all scheduled
activities \"/>\" \
126: "</Command>" \
127: "" \
128: "<!-- Command -->" \
129: "<!-- CommandMsg name= \"PrepareScheduleFile\" id= \"002\" description= \"Prepare an
activities schedule for downlink\" -->" \
130: "<!-- /Command -->" \
131: "" \
132: "</Interface>" \
133: "" \
134: "<Interface id= \"3\" name= \"ActivityManagerStatusInterface \">>\" \
135: "" \
136: "<Variable name= \"ActivityManagerStatus\" kind= \"Status\" format= \"INT16 \">>\" \
137: "<Drange name= \"ActivityManagerStatusEnum \">>\" \
138: "<Option value= \"0\" name= \"NOT_INITIALIZED \"/><!-- The ActivityManager has not been
successfully initialized -->\" \
139: "<Option value= \"1\" name= \"INITIALIZING \"/><!-- The ActivityManager is in the process of
initializing -->\" \

```

140: "<Option value= \"2 \" name= \"RUNNING \"/><!-- The ActivityManager is initialized and is running -->" \

141: "<Option value= \"3 \" name= \"TERMINATING \"/><!-- The ActivityManager is shutting down -->" \

142: "</Drange>" \

143: "</Variable>" \

144: "" \

145: "<Variable name= \"ActivitiesCurrentlyScheduled \" kind= \"tbd \" format= \"UINT16 \"/>" \

146: "<Variable name= \"ActivitiesExecuted \" kind= \"tbd \" format= \"UINT16 \"/>" \

147: "<Variable name= \"ActivitiesExecutedSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \

148: "<Variable name= \"ActivitiesExecutedFailed \" kind= \"tbd \" format= \"UINT16 \"/>" \

149: "<Variable name= \"ActivitiesDeleted \" kind= \"tbd \" format= \"UINT16 \"/>" \

150: "" \

151: "<Variable name= \"ScheduleActivityReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \

152: "<Variable name= \"ScheduleActivityAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \

153: "<Variable name= \"ScheduleActivitySuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \

154: "<Variable name= \"ScheduleActivityFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \

155: "" \

156: "<Variable name= \"UpdateActivityReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \

157: "<Variable name= \"UpdateActivityAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \

158: "<Variable name= \"UpdateActivitySuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \

159: "<Variable name= \"UpdateActivityFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \

160: "" \

161: "<Variable name= \"UpdateActivityStatusReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \

162: "<Variable name= \"UpdateActivityStatusAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \

163: "<Variable name= \"UpdateActivityStatusSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \

164: "<Variable name= \"UpdateActivityStatusFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \

165: "" \

166: "<Variable name= \"DeleteActivityReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \

167: "<Variable name= \"DeleteActivityAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \

168: "<Variable name= \"DeleteActivitySuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \

169: "<Variable name= \"DeleteActivityFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \

170: "" \

171: "<Variable name= \"DeleteAllActivitiesReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \

172: "<Variable name= \"DeleteAllActivitiesAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \

173: "<Variable name= \"DeleteAllActivitiesSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \

174: "<Variable name= \"DeleteAllActivitiesFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \

175: "" \

176: "<!-- Variable name= \"PrepSchedFileReceived \" kind= \"tbd \" format= \"UINT16 \"/ -->" \

177: "<!-- Variable name= \"PrepSchedFileAccepted \" kind= \"tbd \" format= \"UINT16 \"/ -->" \

178: "<!-- Variable name= \"PrepSchedFileSuccess \" kind= \"tbd \" format= \"UINT16 \"/ -->" \


```

179: "<!-- Variable name= \"PrepSchedFileFailure \" kind= \"tbd \" format= \"UINT16 \" / -->" \
180: "" \
181: "<Command>" \
182: "<CommandMsg name= \"SendActivityManagerStatusMsg \" id= \"001 \" description= \"Send the
ActivityManagerStatusMsg DataMsg \"/>" \
183: "</Command>" \
184: "" \
185: "<Notification>" \
186: "<DataMsg name= \"ActivityManagerStatusMsg \" id= \"002 \" msgArrival= \"EVENT \"/>" \
187: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
188: "<VariableRef name= \"ActivityManagerStatus \"/>" \
189: "<VariableRef name= \"ActivitiesCurrentlyScheduled \"/>" \
190: "<VariableRef name= \"ActivitiesExecuted \"/>" \
191: "<VariableRef name= \"ActivitiesExecutedSuccess \"/>" \
192: "<VariableRef name= \"ActivitiesExecutedFailed \"/>" \
193: "<VariableRef name= \"ActivitiesDeleted \"/>" \
194: "<VariableRef name= \"ScheduleActivityReceived \"/>" \
195: "<VariableRef name= \"ScheduleActivityAccepted \"/>" \
196: "<VariableRef name= \"ScheduleActivitySuccess \"/>" \
197: "<VariableRef name= \"ScheduleActivityFailure \"/>" \
198: "<VariableRef name= \"UpdateActivityReceived \"/>" \
199: "<VariableRef name= \"UpdateActivitySuccess \"/>" \
200: "<VariableRef name= \"UpdateActivityFailure \"/>" \
201: "<VariableRef name= \"UpdateActivityStatusReceived \"/>" \
202: "<VariableRef name= \"UpdateActivityStatusAccepted \"/>" \
203: "<VariableRef name= \"UpdateActivityStatusSuccess \"/>" \
204: "<VariableRef name= \"UpdateActivityStatusFailure \"/>" \
205: "<VariableRef name= \"DeleteActivityReceived \"/>" \
206: "<VariableRef name= \"DeleteActivityAccepted \"/>" \
207: "<VariableRef name= \"DeleteActivitySuccess \"/>" \
208: "<VariableRef name= \"DeleteActivityFailure \"/>" \
209: "<VariableRef name= \"DeleteAllActivitiesReceived \"/>" \
210: "<VariableRef name= \"DeleteAllActivitiesAccepted \"/>" \
211: "<VariableRef name= \"DeleteAllActivitiesSuccess \"/>" \
212: "<VariableRef name= \"DeleteAllActivitiesFailure \"/>" \
213: "<!-- VariableRef name= \"PrepSchedFileReceived \" / -->" \
214: "<!-- VariableRef name= \"PrepSchedFileAccepted \" / -->" \
215: "<!-- VariableRef name= \"PrepSchedFileSuccess \" / -->" \
216: "<!-- VariableRef name= \"PrepSchedFileFailure \" / -->" \
217: "</DataMsg>" \
218: "</Notification>" \

```

```

219: "" \
220: "</Interface>" \
221: "" \
222: "<Interface name= \"ICSDebugInterface \" id= \"4 \">>" \
223: "" \
224: "<!--" \
225: "Note: DebugLevel is a bit field with the following assigned values:" \
226: "DEBUG_ENTRY_AND_EXIT = 0x01," \
227: "DEBUG_ENTRY_PARAMETERS = 0x02," \
228: "DEBUG_EXIT_PARAMETERS = 0x04," \
229: "DEBUG_LEVEL_LOW = 0x08," \
230: "DEBUG_LEVEL_MEDIUM = 0x10," \
231: "DEBUG_LEVEL_HIGH = 0x20." \
232: "The values are OR'd to determine the debug information to be logged." \
233: "-->" \
234: "" \
235: "<Variable name= \"DebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
236: "<Variable name= \"CurrentDebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
237: "" \
238: "<Variable name= \"SetDebugLevelReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \
239: "<Variable name= \"SetDebugLevelAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \
240: "<Variable name= \"SetDebugLevelSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \
241: "<Variable name= \"SetDebugLevelFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \
242: "" \
243: "<Command>" \
244: "<CommandMsg name= \"SetDebugLevel \" id= \"001 \" description= \"Set the debug log verbosity level \">>" \
245: "<VariableRef name= \"DebugLevel \"/>" \
246: "</CommandMsg>" \
247: "</Command>" \
248: "" \
249: "<Command>" \
250: "<CommandMsg name= \"SendDebugStatus \" id= \"002 \"/>" \
251: "</Command>" \
252: "" \
253: "<Notification>" \
254: "<DataMsg name= \"DebugStatus \" id= \"003 \" msgArrival= \"EVENT \">>" \
255: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
256: "<VariableRef name= \"CurrentDebugLevel \"/>" \
257: "<VariableRef name= \"SetDebugLevelReceived \"/>" \
258: "<VariableRef name= \"SetDebugLevelAccepted \"/>" \

```

```

259: "<VariableRef name= \"SetDebugLevelSuccess \"/>" \
260: "<VariableRef name= \"SetDebugLevelFailure \"/>" \
261: "</DataMsg>" \
262: "</Notification>" \
263: "" \
264: "</Interface>" \
265: "" \
266: "<Interface name= \"ICSTaskControlInterface \" id= \"5 \"/>" \
267: "" \
268: "<Command>" \
269: "<CommandMsg name= \"DestroyTask \" id= \"001 \"/>" \
270: "</Command>" \
271: "" \
272: "<!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->"
\
273: "<!-- Other possible requests include GetPriority, and GetHeartBeatCount -->" \
274: "" \
275: "</Interface>" \
276: "" \
277: "</xTEDS>" \
278: ""
279:
280: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/StarCamera.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS                xTEDS02.xsd"
name="StarTrackerxTEDS"
4:   version="2.0">
5:   <Device name="TermaHE5ASStarTracker" kind="scam" description="Terma HE-5as Star Camera"
/>
6:
7:   <Interface name="ScamBasic" id="1">
8:     <Qualifier name="headID" value="0"/>
9:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
10:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
11:    <Variable name="AttitudeDev" kind="attitude" format="FLOAT32" length="4"
description="Quaternion describing attitude of the device in inertial space (ECI)" >
12:      <Qualifier name="representation" value="quaternion" />
13:      <Qualifier name="frameFrom" value="ECIMOD" />
14:      <Qualifier name="frameTo" value="DVF" />
15:    </Variable>
16:    <Variable name="AngularRate" kind="attitudeRate" format="FLOAT32" length="3"
units="rad_s" description="Rate of spin about each device axis" >
17:      <Qualifier name="representation" value="vector" />
18:      <Qualifier name="frameMeasured" value="DVF" />
19:      <Qualifier name="frameResolved" value="DVF" />
20:    </Variable>
21:    <Notification>
22:      <DataMsg name="AnglRate" msgArrival="PERIODIC" msgRate="10" id="1">
23:        <Qualifier name="telemetryLevel" value="1"/>
24:        <VariableRef name="Time"/>
25:        <VariableRef name="SubS"/>
26:        <VariableRef name="AngularRate" />
27:      </DataMsg>
28:    </Notification>
29:    <Notification>
30:      <DataMsg name="Attitude" msgArrival="PERIODIC" msgRate="10" id="2">
31:        <Qualifier name="telemetryLevel" value="1"/>
32:        <VariableRef name="Time"/>
33:        <VariableRef name="SubS"/>
34:        <VariableRef name="AttitudeDev" />
```

```

35:     </DataMsg>
36: </Notification>
37: </Interface>
38:
39: <Interface name="DevPwr" id="2">
40:   <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
41:   <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
42:   <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
43:   <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
44:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
45:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
46:   <Variable name="DevPwrState" kind="Power_State" format="UINT08">
47:     <Drange name="DevPwrStateEnum">
48:       <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
49:       <Option name="DevPwrON" value="1" description="Device may draw full power." />
50:     </Drange>
51:   </Variable>
52:   <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
53:     <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
54:       <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
55:       <Option name="DevPwrON" value="1" description="Device may draw full power." />
56:     </Drange>
57:   </Variable>
58:   <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
59:   <Command>
60:     <CommandMsg name="DevPwrSetState" id="1">
61:       <VariableRef name="DevPwrStateSet" />
62:     </CommandMsg>
63:     <FaultMsg name="DevPwrStateNotSet" id="2">
64:       <VariableRef name="Time" />
65:       <VariableRef name="SubS" />
66:       <VariableRef name="DevPwrState" />
67:       <VariableRef name="DevPwrStateSet" />
68:     </FaultMsg>
69:   </Command>
70:   <Notification>
71:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
72:       <Qualifier name="telemetryLevel" value="1"/>
73:       <VariableRef name="Time" />

```

```

74:     <VariableRef name="SubS" />
75:     <VariableRef name="DevPwrState" />
76:     <VariableRef name="DevPwrStateSet" />
77: </DataMsg>
78: </Notification>
79: <Request>
80:   <CommandMsg name="getPowerInMode" id="4" />
81:   <DataReplyMsg name="powerInMode" id="5">
82:     <VariableRef name="modePowers"/>
83:   </DataReplyMsg>
84: </Request>
85: </Interface>
86:
87: <Interface name="CmpSOH" id="3">
88:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
89:   <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
90:   <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
91:   <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
92:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
93:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
94:   <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
95:   <Request>
96:     <CommandMsg name="GetDeviceTemperature" id="1" />
97:     <DataReplyMsg name="DeviceTempReply" id="2">
98:       <VariableRef name="Time" />
99:       <VariableRef name="SubS" />
100:      <VariableRef name="DeviceTemperature"/>
101:    </DataReplyMsg>
102:  </Request>
103: <Notification>
104:   <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
105:     <Qualifier name="telemetryLevel" value="1"/>
106:     <VariableRef name="Time" />
107:     <VariableRef name="SubS" />
108:     <VariableRef name="DeviceTemperature"/>
109:   </DataMsg>
110: </Notification>
111: </Interface>
112: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/SolarArray.h

```
1: #ifndef _SOLARARRAY_XTEDS_H
2: #define _SOLARARRAY_XTEDS_H
3:
4: #define _STRING_SOLARARRAY_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
   name= \"SolarArray_XTEDS \"\" \
8:   "version= \"2.0 \">\" \
9:   "<Device name= \"SolarArray \" kind= \"sa \" description= \"a device that provides electrical power \"
   >\" \
10:   "<Qualifier name= \"Manufacturer \" value= \"MsiFitsArray \"/>\" \
11:   "<Qualifier name= \"Model \" value= \"1.2.3 \"/>\" \
12:   "<Qualifier name= \"SerialNumber \" value= \"90210 \"/>\" \
13:   "</Device>\" \
14:   "" \
15:   "<Interface name= \"SolarArrayInterface \" id= \"1 \">\" \
16:   "<Qualifier name= \"CellType \" value= \"GaAs \"/>\" \
17:   "<Qualifier name= \"BOLPower \" value= \"80 \" units= \"W \"/>\" \
18:   "<Qualifier name= \"PerformanceDegradation \" value= \"5 \" units= \"percent_year \"/>\" \
19:   "<Qualifier name= \"MaxPowerVoltage \" value= \"20 \" units= \"Volts \"/>\" \
20:   "<Qualifier name= \"MaxPowerCurrent \" value= \"10 \" units= \"Amps \"/>\" \
21:   "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
22:   "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
   scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
23:   "<Variable name= \"SATemp \" kind= \"Temperature \" format= \"FLOAT32 \" units= \"degC \"
   description= \"ambient temperature of array \" />\" \
24:   "<Variable name= \"SACurrent \" kind= \"Current \" format= \"FLOAT32 \" units= \"Amps \"
   description= \"charging is positive; discharging is negative \" />\" \
25:   "<Variable name= \"SADesiredCurrent \" kind= \"SetCurrent \" format= \"FLOAT32 \" units=
   \"Amps \" description= \"charging is positive; discharging is negative \" />\" \
26:   "<Variable name= \"SAUnregVoltage \" kind= \"Voltage \" format= \"FLOAT32 \" units= \"Volts \"
   description= \"voltage at battery terminals \" />\" \
27:   "<Variable name= \"SAOperationState \" kind= \"mode \" format= \"UINT08 \">\" \
28:   "<Drange name= \"OpStateEnum \">\" \
29:   "<Option name= \"Offline \" value= \"0 \"/>\" \
30:   "<Option name= \"Online \" value= \"1 \"/>\" \
31:   "</Drange>\" \
32:   "</Variable>\" \
33:   "" \
```

```

34: "<Command>" \
35: "<CommandMsg id= \"1\" name= \"SetOpState \"/>" \
36: "<VariableRef name= \"SAOperationState \"/>" \
37: "</CommandMsg>" \
38: "</Command>" \
39: "" \
40: "<Command>" \
41: "<CommandMsg id= \"2\" name= \"SetCurrentLimitOut \"/>" \
42: "<VariableRef name= \"SADesiredCurrent \"/>" \
43: "</CommandMsg>" \
44: "</Command>" \
45: "" \
46: "<Notification>" \
47: "<DataMsg name= \"ArraySOH\" description= \"state of health\" id= \"3\" msgArrival= \"PERIODIC \"/>" \
48: "<Qualifier name= \"telemetryLevel\" value= \"1 \"/>" \
49: "<Qualifier name= \"Type\" value= \"SOH \"/>" \
50: "<VariableRef name= \"Time \"/>" \
51: "<VariableRef name= \"SubS \"/>" \
52: "<VariableRef name= \"SATemp \"/>" \
53: "<VariableRef name= \"SACurrent \"/>" \
54: "<VariableRef name= \"SADesiredCurrent \"/>" \
55: "<VariableRef name= \"SAUnregVoltage \"/>" \
56: "<VariableRef name= \"SAOperationState \"/>" \
57: "</DataMsg>" \
58: "</Notification>" \
59: "" \
60: "</Interface>" \
61: "" \
62: "<Interface id= \"2\" name= \"DeployerInterface \"/>" \
63: "<!--description= \"Interface for a deployment mechanism - in this case the release for the array. \"-->" \
64: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds \"/>" \
65: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\" scaleFactor= \".0001\" scaleUnits= \"Seconds \"/>" \
66: "<Variable kind= \"mode\" name= \"ArmedState\" format= \"UINT08 \"/>" \
67: "<Drange name= \"ArmedStateEnum \"/>" \
68: "<Option name= \"Safe\" value= \"0 \"/>" \
69: "<Option name= \"Armed\" value= \"1 \"/>" \
70: "</Drange>" \
71: "</Variable>" \
72: "<Variable kind= \"mode\" name= \"DeployedState\" format= \"UINT08 \"/>" \

```



```

73: "<Drange name= \"DeployedStateEnum \"/>" \
74: "<Option name= \"Stowed \" value= \"0 \"/>" \
75: "<Option name= \"Deploying \" value= \"1 \"/>" \
76: "<Option name= \"Deployed \" value= \"2 \"/>" \
77: "</Drange>" \
78: "</Variable>" \
79: "<Command>" \
80: "<CommandMsg id= \"1 \" name= \"ArmOrDisarmRelease \" description= \"Toggles the safe/armed
state of the deployment mechanism \"/>" \
81: "<VariableRef name= \"ArmedState \"/>" \
82: "</CommandMsg>" \
83: "</Command>" \
84: "<Command>" \
85: "<CommandMsg id= \"2 \" name= \"DeployGo \" description= \"Deploy the array \"/>" \
86: "</Command>" \
87: "<Request>" \
88: "<CommandMsg id= \"3 \" name= \"GetDeployState \" />" \
89: "<DataReplyMsg id= \"4 \" name= \"DeployState \"/>" \
90: "<VariableRef name= \"Time \"/>" \
91: "<VariableRef name= \"SubS \"/>" \
92: "<VariableRef name= \"ArmedState \"/>" \
93: "<VariableRef name= \"DeployedState \"/>" \
94: "</DataReplyMsg>" \
95: "</Request>" \
96: "</Interface>" \
97: "" \
98: "<Interface name= \"CmpSafety \" id= \"3 \"/>" \
99: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>" \
100: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>" \
101: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>" \
102: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>" \
103: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
104: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
105: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units=
\"degC \"/>" \
106: "<Request>" \
107: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \"/>" \
108: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \"/>" \
109: "<VariableRef name= \"Time \"/>" \
110: "<VariableRef name= \"SubS \"/>" \
111: "<VariableRef name= \"DeviceTemperature \"/>" \

```

```
112: "</DataReplyMsg>" \
113: "</Request>" \
114: "<Notification>" \
115: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
116: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
117: "<VariableRef name= \"Time \" />" \
118: "<VariableRef name= \"SubS \" />" \
119: "<VariableRef name= \"DeviceTemperature \"/>" \
120: "</DataMsg>" \
121: "</Notification>" \
122: "</Interface>" \
123: "</xTEDS>" \
124: ""
125:
126: #endif
```

File: sdm/app/test/DMTests/xTEDSRegTests/IRU.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS        ../Schema/xTEDS02.xsd"
name="IRUxTeds"
4:   version="2.0">
5:   <Device name="LN200s_IRU" kind="iru" description="Litton LN200s IRU" />
6:
7:   <Interface name="IRUBasic" id="1">
8:     <Qualifier name="headID" value="0"/>
9:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
10:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
11:    <Variable name="AngularRate" kind="attitudeRate" units="rad_s" format="FLOAT32"
length="3" description="Angular rates about each primary axis of the IRU" >
12:      <Qualifier name="representation" value="vector" />
13:      <Qualifier name="frameMeasured" value="DVF" />
14:      <Qualifier name="frameResolved" value="DVF" />
15:    </Variable>
16:    <Notification>
17:      <DataMsg name="AnglRate" msgArrival="PERIODIC" msgRate="1" id="1">
18:        <Qualifier name="telemetryLevel" value="1"/>
19:        <VariableRef name="Time"/>
20:        <VariableRef name="SubS"/>
21:        <VariableRef name="AngularRate" />
22:      </DataMsg>
23:    </Notification>
24:  </Interface>
25:
26:  <Interface name="DevPwr" id="2">
27:    <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
28:    <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
29:    <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
30:    <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
31:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
32:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
33:    <Variable name="DevPwrState" kind="Power_State" format="UINT08">
34:      <Drange name="DevPwrStateEnum">
35:        <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
```

```

36:     <Option name="DevPwrON" value="1" description="Device may draw full power." />
37: </Drange>
38: </Variable>
39: <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
40:     <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
41:     <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
42:     <Option name="DevPwrON" value="1" description="Device may draw full power." />
43: </Drange>
44: </Variable>
45: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
46: <Command>
47:     <CommandMsg name="DevPwrSetState" id="1">
48:         <VariableRef name="DevPwrStateSet" />
49:     </CommandMsg>
50:     <FaultMsg name="DevPwrStateNotSet" id="2">
51:         <VariableRef name="Time" />
52:         <VariableRef name="SubS" />
53:         <VariableRef name="DevPwrState" />
54:         <VariableRef name="DevPwrStateSet" />
55:     </FaultMsg>
56: </Command>
57: <Notification>
58:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
59:         <Qualifier name="telemetryLevel" value="1"/>
60:         <VariableRef name="Time" />
61:         <VariableRef name="SubS" />
62:         <VariableRef name="DevPwrState" />
63:         <VariableRef name="DevPwrStateSet" />
64:     </DataMsg>
65: </Notification>
66: <Request>
67:     <CommandMsg name="getPowerInMode" id="4" />
68:     <DataReplyMsg name="powerInMode" id="5">
69:         <VariableRef name="modePowers"/>
70:     </DataReplyMsg>
71: </Request>
72: </Interface>
73:
74: <Interface name="CmpSOH" id="3">
75:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>

```

```

76: <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
77: <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
78: <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
79: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
80:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
81: <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
82: <Request>
83:     <CommandMsg name="GetDeviceTemperature" id="1" />
84:     <DataReplyMsg name="DeviceTempReply" id="2">
85:         <VariableRef name="Time" />
86:         <VariableRef name="SubS" />
87:         <VariableRef name="DeviceTemperature"/>
88:     </DataReplyMsg>
89: </Request>
90: <Notification>
91:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
92:         <Qualifier name="telemetryLevel" value="1"/>
93:         <VariableRef name="Time" />
94:         <VariableRef name="SubS" />
95:         <VariableRef name="DeviceTemperature"/>
96:     </DataMsg>
97: </Notification>
98: </Interface>
99: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/Targeting.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS    ../Schema/xTEDS02.xsd"
name="Targeting_App_xTeds" version="2.1">
4:   <Application name="TargetingControl" version="1.0" kind="SHAP" description="Calculates current
attitude errors for use by the control law application" />
5:   <Interface name="TargetingInterface" id="1">
6:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
7:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32" scaleFactor=".0001"
scaleUnits="Seconds" />
8:     <Variable name="AttitudeError" kind="attitudeDelta" length="4" format="FLOAT32"
description="Difference between current and desired attitude">
9:       <Qualifier name="representation" value="quaternion"/>
10:      <Qualifier name="frameFrom" value="SVF"/>
11:      <Qualifier name="frameTo" value="SVF"/>
12:    </Variable>
13:    <Variable name="Offset" kind="rollPitchYaw" length="3" units="rad" format="FLOAT32"
description="pushbroom offsets from LVLH [rad]"/>
14:    <Variable name="ADCSMode" kind="mode" format="UINT08" defaultValue="Attitude Control
Mode">
15:      <Drange name="ADCSModeEnum">
16:        <Option name="Standby" value="0"/>
17:        <Option name="Detumble" value="1"/>
18:        <Option name="SunTrack" value="2"/>
19:        <Option name="NadirTrack" value="3"/>
20:        <Option name="TargetTrack" value="4"/>
21:      </Drange>
22:    </Variable>
23:    <Variable name="Target" kind="position" length="3" units="km" format="FLOAT64"
description="ECI position vector of a target at which we wish to point">
24:      <Qualifier name="representation" value="vector"/>
25:      <Qualifier name="frameMeasured" value="ECIMOD"/>
26:      <Qualifier name="frameResolved" value="ECIMOD"/>
27:    </Variable>
28:    <Notification>
29:      <DataMsg name="AttitudeErrorMsg" id="1" msgArrival="PERIODIC" msgRate="100"
description="100HZ broadcast of attitude error">
30:        <VariableRef name="Time"/>
31:        <VariableRef name="SubS"/>
32:        <VariableRef name="AttitudeError"/>
```

```
33:     </DataMsg>
34: </Notification>
35: <Command>
36:   <CommandMsg id="2" name="SetADCMode" description="Command to set the control mode">
37:     <VariableRef name="ADCMode"/>
38:   </CommandMsg>
39: </Command>
40: <Command>
41:   <CommandMsg id="3" name="SetPushBroomOffset" description="Sets the tracking offset">
42:     <VariableRef name="Offset"/>
43:   </CommandMsg>
44: </Command>
45: <Command>
46:   <CommandMsg id="4" name="SetTarget" description="Sets the target at which we wish to
point">
47:     <VariableRef name="Target"/>
48:   </CommandMsg>
49: </Command>
50: </Interface>
51: </xTEDS>
```

File: sdm/app/test/DMTests/xTEDSRegTests/AnglAccelToTorque.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS                                xTEDS02.xsd"
name="Torque_Control_xTeds"
4:   version="2.0">
5:   <Application name="TorqueControl" version="1.0" kind="AHAP" description="Takes commands
for angular acceleration, converts to torque, and distributes torque commands to actuators" />
6:
7:   <Interface name="TorqueControlInterface" id="1">
8:     <Variable name="AnglAccel" kind="angularAcceleration" length="3" format="FLOAT64"
description="Commanded angular acceleration">
9:       <Qualifier name="representation" value="vector"/>
10:      <Qualifier name="frameMeasured" value="SVF"/>
11:      <Qualifier name="frameResolved" value="SVF"/>
12:    </Variable>
13:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
14:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
15:    <Variable kind="Torque" name="X" units="nm" format="FLOAT32" description="X component
of torque to apply" />
16:    <Variable kind="Torque" name="Y" units="nm" format="FLOAT32" description="Y component
of torque to apply" />
17:    <Variable kind="Torque" name="Z" units="nm" format="FLOAT32" description="Z component
of torque to apply" />
18:    <Command>
19:      <CommandMsg id="1" name="SCAngularAccelCmd" description="Command to apply angular
acceleration to the spacecraft">
20:        <VariableRef name="AnglAccel"/>
21:      </CommandMsg>
22:    </Command>
23:    <Notification>
24:      <DataMsg id="2" name="SVTorqueToApply" description="Torque equivalent to requested
angular acceleration" msgArrival="PERIODIC" msgRate="10">
25:        <VariableRef name="Time"/>
26:        <VariableRef name="SubS"/>
27:        <VariableRef name="X"/>
28:        <VariableRef name="Y"/>
29:        <VariableRef name="Z"/>
30:      </DataMsg>
31:    </Notification>
```


32: </Interface>

33: </xTEDS>

File: sdm/app/test/DMTests/xTEDSRegTests/GPS.h

```
1: #ifndef _BASICGPSRECEIVER_XTEDS_H
2: #define _BASICGPSRECEIVER_XTEDS_H
3:
4: #define _STRING_BASICGPSRECEIVER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \" \
name= \"BasicGPSxTEDS \"\" \
8:   \"version= \"2.0\">\" \
9:   \"<Device name= \"BasicGPSReceiver \" kind= \"sgr \" description= \"A GPS receiver that produces
pseudorange measurements only \" />\" \
10:   \"<Interface name= \"GPSBasic \" id= \"1\">\" \
11:   \"<Qualifier name= \"headID \" value= \"0\" />\" \
12:   \"<Variable name= \"numVisibleSats \" kind= \"listSize \" format= \"INT16 \" description= \"Number
of GPS satellites visible in this observation \" />\" \
13:   \"<Variable name= \"recvTime \" kind= \"time \" format= \"INT32 \" units= \"s \" description=
\"Receiver clock time since GPS epoch \">\" \
14:   \"<Qualifier name= \"timeFrame \" value= \"GPST1 \" />\" \
15:   \"</Variable>\" \
16:   \"<Variable name= \"PRNs \" kind= \"ID \" format= \"INT16 \" length= \"28 \" description= \"list of
PRN numbers of the visible GPS satellites \">\" \
17:   \"<Qualifier name= \"representation \" value= \"array \" />\" \
18:   \"</Variable>\" \
19:   \"<Variable name= \"PRN \" kind= \"ID \" format= \"INT16 \" description= \"PRN of a single GPS sat
generating a NAV message \" />\" \
20:   \"<Variable name= \"Prange \" kind= \"distance \" format= \"FLOAT32 \" length= \"28 \" units= \"km
\" description= \"Pseudorange measurements for each visible GPS satellite \">\" \
21:   \"<Qualifier name= \"representation \" value= \"array \" />\" \
22:   \"</Variable>\" \
23:   \"<Variable name= \"epochTime \" kind= \"epoch \" format= \"INT16 \" length= \"6 \" description=
\"YMDHMS definition of the current GPS epoch time \">\" \
24:   \"<Qualifier name= \"representation \" value= \"array \" />\" \
25:   \"</Variable>\" \
26:   \"<Variable name= \"nav \" kind= \"navMessage \" format= \"FLOAT32 \" length= \"28 \" \
description= \"Nav message from this satellite: ephemeris, clock corrections \">\" \
27:   \"<Qualifier name= \"representation \" value= \"array \" />\" \
28:   \"</Variable>\" \
29:   \"<Notification>\" \
30:   \"<DataMsg name= \"GPSSatPrange \" msgArrival= \"PERIODIC \" msgRate= \"10 \" id= \"1\">\" \
31:   \"<Qualifier name= \"telemetryLevel \" value= \"1\" />\" \
```

```

32: "<VariableRef name= \"numVisibleSats \" />" \
33: "<VariableRef name= \"recvTime \" />" \
34: "<VariableRef name= \"PRNs \" />" \
35: "<VariableRef name= \"Prange \" />" \
36: "</DataMsg>" \
37: "</Notification>" \
38: "<Notification>" \
39: "<DataMsg name= \"GPSNavMessage \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"2 \">>" \
40: "<VariableRef name= \"PRN \" />" \
41: "<VariableRef name= \"epochTime \" />" \
42: "<VariableRef name= \"nav \" />" \
43: "</DataMsg>" \
44: "</Notification>" \
45: "</Interface>" \
46: "<Interface name= \"DevPwr \" id= \"2 \">>" \
47: "<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \"/>" \
48: "<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \"/>" \
49: "<Qualifier name= \"CurrentHiWarning \" value= \"3.5 \" units= \"A \"/>" \
50: "<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \"/>" \
51: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
52: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
53: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \">>" \
54: "<Drange name= \"DevPwrStateEnum \">>" \
55: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />" \
56: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
57: "</Drange>" \
58: "</Variable>" \
59: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \">>" \
60: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to DevPwrStateEnumeration. \" />" \
61: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />" \
62: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
63: "</Drange>" \
64: "</Variable>" \
65: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length= \"2 \" />" \
66: "<Command>" \
67: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \">>" \
68: "<VariableRef name= \"DevPwrStateSet \" />" \

```

69: "</CommandMsg>" \
 70: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \">" \
 71: "<VariableRef name= \"Time \" />" \
 72: "<VariableRef name= \"SubS \" />" \
 73: "<VariableRef name= \"DevPwrState \" />" \
 74: "<VariableRef name= \"DevPwrStateSet \" />" \
 75: "</FaultMsg>" \
 76: "</Command>" \
 77: "<Notification>" \
 78: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">" \
 79: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
 80: "<VariableRef name= \"Time \" />" \
 81: "<VariableRef name= \"SubS \" />" \
 82: "<VariableRef name= \"DevPwrState \" />" \
 83: "<VariableRef name= \"DevPwrStateSet \" />" \
 84: "</DataMsg>" \
 85: "</Notification>" \
 86: "<Request>" \
 87: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
 88: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">" \
 89: "<VariableRef name= \"modePowers \"/>" \
 90: "</DataReplyMsg>" \
 91: "</Request>" \
 92: "</Interface>" \
 93: "" \
 94: "<Interface name= \"CmpSOH \" id= \"3 \">" \
 95: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>" \
 96: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>" \
 97: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>" \
 98: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>" \
 99: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
 100: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
 101: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \" />" \
 102: "<Request>" \
 103: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />" \
 104: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">" \
 105: "<VariableRef name= \"Time \" />" \
 106: "<VariableRef name= \"SubS \" />" \
 107: "<VariableRef name= \"DeviceTemperature \"/>" \

```
108: "</DataReplyMsg>" \
109: "</Request>" \
110: "<Notification>" \
111: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">>" \
112: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
113: "<VariableRef name= \"Time \" />" \
114: "<VariableRef name= \"SubS \" />" \
115: "<VariableRef name= \"DeviceTemperature \"/>" \
116: "</DataMsg>" \
117: "</Notification>" \
118: "</Interface>" \
119: "</xTEDS>" \
120: ""
121:
122: #endif
```

File: sdm/app/test/DMTests/xTEDSRegTests/ActivityAgent.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd"
name="ActivityAgentXTEDS" description="ActivityAgent xTEDS" version="2.4">
4:
5:     <Application name="ActivityAgent" kind="AutonomyFlightSoftware" description="Autonomous
Tasking Executive (ATE), ActivityAgent"/>
6:
7:     <!-- Note: An ActivityAgent must implement the ActivityInterface and should implement the
ActivityAgentStatusInterface. -->
8:
9:     <Interface name="ActivityInterface" id="1"><!-- This is a template for an ActivityAgent Interface.
Unique activity parameters must be added as necessary. -->
10:
11:         <Variable name="ActivityName" kind="String" format="INT08" length="33"/><!-- 33-byte, null
terminated string -->
12:         <Variable name="ActivityId" kind="ID" format="UINT32"/>
13:         <Variable name="ActivityStatus" kind="Status" format="INT16">
14:             <Drange name="ActivityStatusEnum">
15:                 <Option value="0" name="SCHEDULE_FAILURE"/><!-- The activity could not be scheduled
as requested -->
16:                 <Option value="1" name="WAITING"/><!-- The activity has been inserted into the schedule -->
17:                 <Option value="2" name="ENABLED"/><!-- The activity has been sent a command to execute -
->
18:                 <Option value="3" name="TERMINATED"/><!-- The activity has been sent a command to
abort -->
19:                 <Option value="4" name="EXECUTING"/><!-- The activity is currently executing -->
20:                 <Option value="5" name="DONE_FAILURE"/><!-- The activity has completed abnormally -->
21:                 <Option value="6" name="DONE_SUCCESS"/><!-- The activity has completed normally -->
22:                 <Option value="7" name="DONE_NOT_EXECUTED"/><!-- The activity has terminated
without executing -->
23:             </Drange>
24:         </Variable>
25:
26:     <!-- Add Variables for the unique activity parameters -->
27:
28:     <Command> <!-- Provides the capability to request an Activity from the ground -->
29:         <CommandMsg name="RequestActivityCmd" id="001" description="Determine activity state
variables and request to be scheduled">
```

```

30:     <VariableRef name="ActivityId"/><!-- Use 0 for on-board requests. The ActivityId will then be
assigned by the ActivityManager -->
31:
32:     <!-- Add VariableRefs for the unique activity parameters -->
33:
34: </CommandMsg>
35: </Command>
36:
37: <Request>
38:     <CommandMsg name="RequestActivity" id="002" description="Determine activity state
variables and request to be scheduled">
39:     <VariableRef name="ActivityId"/><!-- Use 0 for on-board requests. The ActivityId will then be
assigned by the ActivityManager -->
40:
41:     <!-- Add VariableRefs for the unique activity parameters -->
42:
43: </CommandMsg>
44: <DataReplyMsg name="RequestActivityReply" id="003">
45:     <VariableRef name="ActivityId"/>
46:     <VariableRef name="ActivityStatus"/>
47: </DataReplyMsg>
48: </Request>
49:
50: <Command> <!-- Provides the capability to update an Activity from the ground -->
51:     <CommandMsg name="UpdateRequestCmd" id="004" description="Update activity state
variables and request a schedule update if necessary">
52:     <VariableRef name="ActivityId"/>
53:
54:     <!-- Add VariableRefs for the unique activity parameters -->
55:
56: </CommandMsg>
57: </Command>
58:
59: <Request>
60:     <CommandMsg name="UpdateRequest" id="005" description="Update activity state variables
and request a schedule update if necessary">
61:     <VariableRef name="ActivityId"/>
62:
63:     <!-- Add VariableRefs for the unique activity parameters -->
64:
65: </CommandMsg>
66: <DataReplyMsg name="UpdateRequestReply" id="006">

```

```

67:     <VariableRef name="ActivityId"/>
68:     <VariableRef name="ActivityStatus"/>
69: </DataReplyMsg>
70: </Request>
71:
72: <Command>
73:     <CommandMsg name="Reschedule" id="007" description="The activity has been removed from
the schedule and needs to be rescheduled">
74:         <VariableRef name="ActivityId"/>
75:     </CommandMsg>
76: </Command>
77:
78: <Command>
79:     <CommandMsg name="Delete" id="008" description="Delete the activity">
80:         <VariableRef name="ActivityId"/>
81:     </CommandMsg>
82: </Command>
83:
84: <Command>
85:     <CommandMsg name="Execute" id="009" description="Execute the activity">
86:         <VariableRef name="ActivityId"/>
87:     </CommandMsg>
88: </Command>
89:
90: <Command>
91:     <CommandMsg name="Abort" id="010" description="Abort execution of the activity">
92:         <VariableRef name="ActivityId"/>
93:     </CommandMsg>
94: </Command>
95:
96: <Command>
97:     <CommandMsg name="SendActivityStatusMsg" id="011" description="Send the
ActivityStatusMsg DataMsg">
98:         <VariableRef name="ActivityId"/>
99:     </CommandMsg>
100: </Command>
101:
102: <Notification>
103:     <DataMsg name="ActivityStatusMsg" id="012" msgArrival="EVENT">
104:         <Qualifier value="1" name="telemetryLevel"/>
105:         <VariableRef name="ActivityId"/>

```



```

106:     <VariableRef name="ActivityName"/>
107:     <VariableRef name="ActivityStatus"/>
108: </DataMsg>
109: </Notification>
110:
111: </Interface>
112:
113: <Interface name="ActivityAgentStatusInterface" id="2">
114:
115:     <Variable name="ActivityAgentStatus" kind="Status" format="INT16">
116:         <Drange name="ActivityAgentStatusEnum">
117:             <Option value="0" name="NOT_INITIALIZED"/><!-- The ActivityAgent has not been
successfully initialized -->
118:             <Option value="1" name="INITIALIZING"/><!-- The ActivityAgent is in the process of
initializing -->
119:             <Option value="2" name="RUNNING"/><!-- The ActivityAgent is initialized and is running --
>
120:             <Option value="3" name="TERMINATING"/><!-- The ActivityAgent is shutting down -->
121:         </Drange>
122:     </Variable>
123:
124:     <Variable name="ActivityAgentName" kind="tbd" format="UINT08" length="33"/><!-- 33-byte,
null terminated string -->
125:
126:     <Variable name="ActivitiesCurrentlyScheduled" kind="tbd" format="UINT16"/>
127:
128:     <Variable name="ActivitiesExecuted" kind="tbd" format="UINT16"/>
129:     <Variable name="ActivitiesExecutedSuccess" kind="tbd" format="UINT16"/>
130:     <Variable name="ActivitiesExecutedFailure" kind="tbd" format="UINT16"/>
131:     <Variable name="ActivitiesDeleted" kind="tbd" format="UINT16"/>
132:
133:     <Variable name="RequestActivityReceived" kind="tbd" format="UINT16"/>
134:     <Variable name="RequestActivityAccepted" kind="tbd" format="UINT16"/>
135:     <Variable name="RequestActivitySuccess" kind="tbd" format="UINT16"/>
136:     <Variable name="RequestActivityFailure" kind="tbd" format="UINT16"/>
137:
138:     <Variable name="UpdateRequestReceived" kind="tbd" format="UINT16"/>
139:     <Variable name="UpdateRequestAccepted" kind="tbd" format="UINT16"/>
140:     <Variable name="UpdateRequestSuccess" kind="tbd" format="UINT16"/>
141:     <Variable name="UpdateRequestFailure" kind="tbd" format="UINT16"/>
142:
143:     <Variable name="ExecuteReceived" kind="tbd" format="UINT16"/>

```

```

144: <Variable name="ExecuteAccepted" kind="tbd" format="UINT16"/>
145: <Variable name="ExecuteSuccess" kind="tbd" format="UINT16"/>
146: <Variable name="ExecuteFailure" kind="tbd" format="UINT16"/>
147:
148: <Variable name="RescheduleReceived" kind="tbd" format="UINT16"/>
149: <Variable name="RescheduleAccepted" kind="tbd" format="UINT16"/>
150: <Variable name="RescheduleSuccess" kind="tbd" format="UINT16"/>
151: <Variable name="RescheduleFailure" kind="tbd" format="UINT16"/>
152:
153: <Variable name="DeleteReceived" kind="tbd" format="UINT16"/>
154: <Variable name="DeleteAccepted" kind="tbd" format="UINT16"/>
155: <Variable name="DeleteSuccess" kind="tbd" format="UINT16"/>
156: <Variable name="DeleteFailure" kind="tbd" format="UINT16"/>
157:
158: <Command>
159:     <CommandMsg name="SendActivityAgentStatusMsg" id="001" description="Send the
ActivityAgentStatusMsg DataMsg"/>
160: </Command>
161:
162: <Notification>
163:     <DataMsg name="ActivityAgentStatusMsg" id="002" msgArrival="EVENT">
164:         <Qualifier value="1" name="telemetryLevel"/>
165:         <VariableRef name="ActivityAgentStatus"/>
166:         <VariableRef name="ActivityAgentName"/>
167:         <VariableRef name="ActivitiesCurrentlyScheduled"/>
168:         <VariableRef name="ActivitiesExecuted"/>
169:         <VariableRef name="ActivitiesExecutedSuccess"/>
170:         <VariableRef name="ActivitiesExecutedFailure"/>
171:         <VariableRef name="ActivitiesDeleted"/>
172:         <VariableRef name="RequestActivityReceived"/>
173:         <VariableRef name="RequestActivityAccepted"/>
174:         <VariableRef name="RequestActivitySuccess"/>
175:         <VariableRef name="RequestActivityFailure"/>
176:         <VariableRef name="UpdateRequestReceived"/>
177:         <VariableRef name="UpdateRequestAccepted"/>
178:         <VariableRef name="UpdateRequestSuccess"/>
179:         <VariableRef name="UpdateRequestFailure"/>
180:         <VariableRef name="ExecuteReceived"/>
181:         <VariableRef name="ExecuteAccepted"/>
182:         <VariableRef name="ExecuteSuccess"/>
183:         <VariableRef name="ExecuteFailure"/>

```

```

184:     <VariableRef name="RescheduleReceived"/>
185:     <VariableRef name="RescheduleAccepted"/>
186:     <VariableRef name="RescheduleSuccess"/>
187:     <VariableRef name="RescheduleFailure"/>
188:     <VariableRef name="DeleteReceived"/>
189:     <VariableRef name="DeleteAccepted"/>
190:     <VariableRef name="DeleteSuccess"/>
191:     <VariableRef name="DeleteFailure"/>
192: </DataMsg>
193: </Notification>
194:
195: </Interface>
196:
197: <Interface name="ATEDebugInterface" id="3">
198:
199:     <!--
200:     Note: DebugLevel is a bit field with the following assigned values:
201:         DEBUG_ENTRY_AND_EXIT   = 0x01,
202:         DEBUG_ENTRY_PARAMETERS = 0x02,
203:         DEBUG_EXIT_PARAMETERS  = 0x04,
204:         DEBUG_LEVEL_LOW        = 0x08,
205:         DEBUG_LEVEL_MEDIUM     = 0x10,
206:         DEBUG_LEVEL_HIGH       = 0x20.
207:     The values are OR'd to determine the debug information to be logged.
208:     -->
209:
210:     <Variable name="DebugLevel" kind="tbd" format="UINT16"/>
211:     <Variable name="CurrentDebugLevel" kind="tbd" format="UINT16"/>
212:
213:     <Variable name="SetDebugLevelReceived" kind="tbd" format="UINT16"/>
214:     <Variable name="SetDebugLevelAccepted" kind="tbd" format="UINT16"/>
215:     <Variable name="SetDebugLevelSuccess" kind="tbd" format="UINT16"/>
216:     <Variable name="SetDebugLevelFailure" kind="tbd" format="UINT16"/>
217:
218:     <Command>
219:         <CommandMsg name="SetDebugLevel" id="001" description="Set the debug log verbosity
level">
220:             <VariableRef name="DebugLevel"/>
221:         </CommandMsg>
222:     </Command>
223:

```

```

224: <Command>
225:   <CommandMsg name="SendDebugStatus" id="002"/>
226: </Command>
227:
228: <Notification>
229:   <DataMsg name="DebugStatus" id="003" msgArrival="EVENT">
230:     <Qualifier value="1" name="telemetryLevel"/>
231:     <VariableRef name="CurrentDebugLevel"/>
232:     <VariableRef name="SetDebugLevelReceived"/>
233:     <VariableRef name="SetDebugLevelAccepted"/>
234:     <VariableRef name="SetDebugLevelSuccess"/>
235:     <VariableRef name="SetDebugLevelFailure"/>
236:   </DataMsg>
237: </Notification>
238:
239: </Interface>
240:
241: <Interface name="TaskControlInterface" id="4">
242:
243:   <Command>
244:     <CommandMsg name="DestroyTask" id="001"/>
245:   </Command>
246:
247:   <!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->
248:   <!-- Other possible requests include GetPriority, and GetHeartBeatCount -->
249:
250: </Interface>
251:
252: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/Targeting.h

```
1: #ifndef _TARGETINGCONTROL_XTEDS_H
2: #define _TARGETINGCONTROL_XTEDS_H
3:
4: #define _STRING_TARGETINGCONTROL_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"Targeting_App_xTeds \" version= \"2.1 \">\" \
8:   \"<Application name= \"TargetingControl \" version= \"1.0 \" kind= \"SHAP \" description=
\"Calculates current attitude errors for use by the control law application \" />\" \
9:   \"<Interface name= \"TargetingInterface \" id= \"1 \">\" \
10:  \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
11:  \"<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \" />\" \
12:  \"<Variable name= \"AttitudeError \" kind= \"attitudeDelta \" length= \"4 \" format= \"FLOAT32 \"
description= \"Difference between current and desired attitude \">\" \
13:  \"<Qualifier name= \"representation \" value= \"quaternion \" />\" \
14:  \"<Qualifier name= \"frameFrom \" value= \"SVF \" />\" \
15:  \"<Qualifier name= \"frameTo \" value= \"SVF \" />\" \
16:  \"</Variable>\" \
17:  \"<Variable name= \"Offset \" kind= \"rollPitchYaw \" length= \"3 \" units= \"rad \" format=
\"FLOAT32 \" description= \"pushbroom offsets from LVLH [rad] \" />\" \
18:  \"<Variable name= \"ADCSMMode \" kind= \"mode \" format= \"UINT08 \" defaultValue= \"Attitude
Control Mode \">\" \
19:  \"<Drange name= \"ADCSMModeEnum \">\" \
20:  \"<Option name= \"Standby \" value= \"0 \" />\" \
21:  \"<Option name= \"Detumble \" value= \"1 \" />\" \
22:  \"<Option name= \"SunTrack \" value= \"2 \" />\" \
23:  \"<Option name= \"NadirTrack \" value= \"3 \" />\" \
24:  \"<Option name= \"TargetTrack \" value= \"4 \" />\" \
25:  \"</Drange>\" \
26:  \"</Variable>\" \
27:  \"<Variable name= \"Target \" kind= \"position \" length= \"3 \" units= \"km \" format= \"FLOAT64 \"
description= \"ECI position vector of a target at which we wish to point \">\" \
28:  \"<Qualifier name= \"representation \" value= \"vector \" />\" \
29:  \"<Qualifier name= \"frameMeasured \" value= \"ECIMOD \" />\" \
30:  \"<Qualifier name= \"frameResolved \" value= \"ECIMOD \" />\" \
31:  \"</Variable>\" \
32:  \"<Notification>\" \
```

```

33: "<DataMsg name= \"AttitudeErrorMsg \" id= \"1 \" msgArrival= \"PERIODIC \" msgRate= \"100 \"
description= \"100HZ broadcast of attitude error \">" \
34: "<VariableRef name= \"Time \"/>" \
35: "<VariableRef name= \"SubS \"/>" \
36: "<VariableRef name= \"AttitudeError \"/>" \
37: "</DataMsg>" \
38: "</Notification>" \
39: "<Command>" \
40: "<CommandMsg id= \"2 \" name= \"SetADCSCMode \" description= \"Command to set the control
mode \">" \
41: "<VariableRef name= \"ADCSCMode \"/>" \
42: "</CommandMsg>" \
43: "</Command>" \
44: "<Command>" \
45: "<CommandMsg id= \"3 \" name= \"SetPushBroomOffset \" description= \"Sets the tracking offset
\">" \
46: "<VariableRef name= \"Offset \"/>" \
47: "</CommandMsg>" \
48: "</Command>" \
49: "<Command>" \
50: "<CommandMsg id= \"4 \" name= \"SetTarget \" description= \"Sets the target at which we wish to
point \">" \
51: "<VariableRef name= \"Target \"/>" \
52: "</CommandMsg>" \
53: "</Command>" \
54: "</Interface>" \
55: "</xTEDS>" \
56: ""
57:
58: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/IDS.h

```
1: #ifndef _INTELLIGENTDATASTORE_XTEDS_H
2: #define _INTELLIGENTDATASTORE_XTEDS_H
3:
4: #define _STRING_INTELLIGENTDATASTORE_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6: "<xTEDS name= \"IDSxTEDS\" \"\" \
7: "xmlns= \"http://www.interfacecontrol.com/SPA/xTEDS\" \"\" \
8: "xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" \"\" \
9: "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd\" \"\" \
10: "version= \"2.0\">\" \
11: "\" \
12: "<Application name= \"IntelligentDataStore\" kind= \"Software\">\" \
13: "<Qualifier name= \"author\" value= \"Ken_Center\"/>\" \
14: "<Qualifier name= \"company\" value= \"Design_Net_Engineering\"/>\" \
15: "</Application>\" \
16: "\" \
17: "<Interface name= \"FileAccessInterface\" id= \"1\">\" \
18: "<Variable name= \"FileHandle\" format= \"UINT16\" kind= \"Handle\" description= \"This field is
used to identify file requests.\"/>\" \
19: "<Variable name= \"PathName\" format= \"INT08\" kind= \"PathName_String\" length= \"80\"
description= \"This field is the null-terminated path to the file.\"/>\" \
20: "<Variable name= \"FileName\" format= \"INT08\" kind= \"FileName_String\" length= \"80\"
description= \"This field is the null-terminated filename of the request.\"/>\" \
21: "<Variable name= \"FileBuffer\" format= \"UINT08\" kind= \"File_Section\" length= \"8047\"
description= \"This field is the buffer containing the portion of the file requested.\"/>\" \
22: "<Variable name= \"StatusCode\" format= \"UINT08\" kind= \"Status_Code\" description= \"This
field is a status corresponding to a fault.\">\" \
23: "<Drange name= \"StatusCodeTypes\">\" \
24: "<Option name= \"OperationOK\" value= \"1\"/>\" \
25: "<Option name= \"InvalidHandle\" value= \"2\"/>\" \
26: "<Option name= \"FileNotAvailable\" value= \"3\"/>\" \
27: "<Option name= \"InvalidOffset\" value= \"4\"/>\" \
28: "<Option name= \"CouldNotObtainHandle\" value= \"5\"/>\" \
29: "</Drange>\" \
30: "</Variable>\" \
31: "<Variable name= \"FileFlag\" format= \"UINT08\" kind= \"Flag\" description= \"This field
represents the type of handle to obtain.\">\" \
32: "<Drange name= \"FlagTypes\">\" \
33: "<Option name= \"ReadOnly\" value= \"1\"/>\" \
34: "<Option name= \"WriteOnly\" value= \"2\"/>\" \
```

```

35: "<Option name= \"ReadWrite \" value= \"3 \"/>\" \
36: "</Drange>\" \
37: "</Variable>\" \
38: "<Variable name= \"ByteOffset \" format= \"UINT32 \" kind= \"Offset \" description= \"This field is
the byte offset for reading and writing to files. \"/>\" \
39: "<Variable name= \"Length \" format= \"UINT32 \" kind= \"Length \" description= \"This field is the
number of bytes to read/write from the offset. \"/>\" \
40: "<Request>\" \
41: "<CommandMsg name= \"OpenFileHandle \" id= \"1 \"/>\" \
42: "<VariableRef name= \"PathName \"/>\" \
43: "<VariableRef name= \"FileName \"/>\" \
44: "<VariableRef name= \"FileFlag \"/>\" \
45: "</CommandMsg>\" \
46: "<DataReplyMsg name= \"OpenHandleReply \" id= \"2 \"/>\" \
47: "<VariableRef name= \"PathName \"/>\" \
48: "<VariableRef name= \"FileName \"/>\" \
49: "<VariableRef name= \"StatusCode \"/>\" \
50: "<VariableRef name= \"FileHandle \"/>\" \
51: "</DataReplyMsg>\" \
52: "</Request>\" \
53: "<Command>\" \
54: "<CommandMsg name= \"CloseFileHandle \" id= \"3 \"/>\" \
55: "<VariableRef name= \"FileHandle \"/>\" \
56: "</CommandMsg>\" \
57: "<FaultMsg name= \"CloseFileHandleError \" id= \"4 \"/>\" \
58: "<VariableRef name= \"FileHandle \"/>\" \
59: "<VariableRef name= \"StatusCode \"/>\" \
60: "</FaultMsg>\" \
61: "</Command>\" \
62: "<Request>\" \
63: "<CommandMsg name= \"ReadPortion \" id= \"5 \"/>\" \
64: "<VariableRef name= \"FileHandle \"/>\" \
65: "<VariableRef name= \"ByteOffset \"/>\" \
66: "<VariableRef name= \"Length \"/>\" \
67: "</CommandMsg>\" \
68: "<DataReplyMsg name= \"ReadReply \" id= \"6 \"/>\" \
69: "<VariableRef name= \"FileHandle \"/>\" \
70: "<VariableRef name= \"StatusCode \"/>\" \
71: "<VariableRef name= \"Length \"/>\" \
72: "<VariableRef name= \"FileBuffer \"/>\" \
73: "</DataReplyMsg>\" \

```



```

74: "</Request>" \
75: "<Request>" \
76: "<CommandMsg name= \"WritePortion \" id= \"7 \">" \
77: "<VariableRef name= \"FileHandle \"/>" \
78: "<VariableRef name= \"ByteOffset \"/>" \
79: "<VariableRef name= \"Length \"/>" \
80: "<VariableRef name= \"FileBuffer \"/>" \
81: "</CommandMsg>" \
82: "<DataReplyMsg name= \"WriteReply \" id= \"8 \">" \
83: "<VariableRef name= \"FileHandle \"/>" \
84: "<VariableRef name= \"StatusCode \"/>" \
85: "<VariableRef name= \"Length \"/>" \
86: "</DataReplyMsg>" \
87: "</Request>" \
88: "</Interface>" \
89: "" \
90: "<Interface name= \"FileManagementInterface \" id= \"2 \">" \
91: "<Variable name= \"FileListTextBuffer \" format= \"UINT08 \" length= \"8047 \" kind= \"Text_Buffer \" description= \"Text buffer containing File Listing \" />" \
92: "<Variable name= \"FileName \" format= \"INT08 \" length= \"80 \" kind= \"FileName_String \" description= \"Name of File \" />" \
93: "<Variable name= \"PathName \" format= \"INT08 \" length= \"80 \" kind= \"PathName_String \" description= \"File Path Text \" />" \
94: "<Variable name= \"FromFileName \" format= \"INT08 \" length= \"80 \" kind= \"FileName_String \" description= \"Name of Originating File \" />" \
95: "<Variable name= \"FromPathName \" format= \"INT08 \" length= \"80 \" kind= \"PathName_String \" description= \"Originating File Path Text \" />" \
96: "<Variable name= \"ToFileName \" format= \"INT08 \" length= \"80 \" kind= \"FileName_String \" description= \"Name of Destination File \" />" \
97: "<Variable name= \"ToPathName \" format= \"INT08 \" length= \"80 \" kind= \"PathName_String \" description= \"Destination File Path Text \" />" \
98: "<Variable name= \"FileSize \" format= \"UINT32 \" kind= \"File_Size_Bytes \" description= \"Size of File in Bytes \" />" \
99: "<Variable name= \"FileOwner \" format= \"UINT08 \" length= \"16 \" kind= \"File_Owner \" description= \"File owner text string \" />" \
100: "<Variable name= \"FileLocked \" format= \"UINT08 \" kind= \"File_Lock_State \" description= \"Indicates whether file is in use - 1 is available, 2 is locked \" />" \
101: "<Variable name= \"FilePermissions \" format= \"UINT08 \" length= \"16 \" kind= \"File_Permissions \" description= \"File permissions text string \" />" \
102: "<Variable name= \"StatusCode \" format= \"UINT08 \" kind= \"Status_Code \" description= \"Fail code for file management actions \">" \
103: "<Drange name= \"StatusCodeTypes \">" \
104: "<Option name= \"OperationOK \" value= \"1 \"/>" \

```

```

105: "<Option name= \"InvalidPath \" value= \"2 \"/>" \
106: "<Option name= \"InvalidFile \" value= \"3 \"/>" \
107: "<Option name= \"PathExists \" value= \"4 \"/>" \
108: "<Option name= \"FileExists \" value= \"5 \"/>" \
109: "<Option name= \"InvalidPathAndFile \" value= \"6 \"/>" \
110: "<Option name= \"InsufficientMemory \" value= \"7 \"/>" \
111: "<Option name= \"PathNotEmpty \" value= \"8 \"/>" \
112: "</Drange>" \
113: "</Variable>" \
114: "<Variable name= \"FileListCode \" format= \"UINT08 \" kind= \"File_List_Code \" description=
\"Type of File Listing - brief or complete \"/>" \
115: "<Drange name= \"FileListCodeTypes \"/>" \
116: "<Option name= \"Brief \" value= \"1 \"/>" \
117: "<Option name= \"Complete \" value= \"2 \"/>" \
118: "</Drange>" \
119: "</Variable>" \
120: "" \
121: "<Request>" \
122: "<CommandMsg name= \"FileList \" id= \"1 \"/>" \
123: "<VariableRef name= \"PathName \"/>" \
124: "<VariableRef name= \"FileListCode \"/>" \
125: "</CommandMsg>" \
126: "<DataReplyMsg name= \"FileListReply \" id= \"2 \"/>" \
127: "<VariableRef name= \"PathName \"/>" \
128: "<VariableRef name= \"StatusCode \"/>" \
129: "<VariableRef name= \"FileListTextBuffer \"/>" \
130: "</DataReplyMsg>" \
131: "</Request>" \
132: "<Request>" \
133: "<CommandMsg name= \"FileInfo \" id= \"3 \"/>" \
134: "<VariableRef name= \"PathName \"/>" \
135: "<VariableRef name= \"FileName \"/>" \
136: "</CommandMsg>" \
137: "<DataReplyMsg name= \"FileInfoReply \" id= \"4 \"/>" \
138: "<VariableRef name= \"PathName \"/>" \
139: "<VariableRef name= \"FileName \"/>" \
140: "<VariableRef name= \"StatusCode \"/>" \
141: "<VariableRef name= \"FileSize \"/>" \
142: "<VariableRef name= \"FileOwner \"/>" \
143: "<VariableRef name= \"FileLocked \"/>" \
144: "<VariableRef name= \"FilePermissions \"/>" \

```

145: "</DataReplyMsg>" \
 146: "</Request>" \
 147: "<Request>" \
 148: "<CommandMsg name= \"FileCopy\" id= \"5\">" \
 149: "<VariableRef name= \"FromPathName\"/>" \
 150: "<VariableRef name= \"FromFileName\"/>" \
 151: "<VariableRef name= \"ToPathName\"/>" \
 152: "<VariableRef name= \"ToFileName\"/>" \
 153: "</CommandMsg>" \
 154: "<DataReplyMsg name= \"FileCopyReply\" id= \"6\">" \
 155: "<VariableRef name= \"FromPathName\"/>" \
 156: "<VariableRef name= \"FromFileName\"/>" \
 157: "<VariableRef name= \"ToPathName\"/>" \
 158: "<VariableRef name= \"ToFileName\"/>" \
 159: "<VariableRef name= \"StatusCode\"/>" \
 160: "</DataReplyMsg>" \
 161: "</Request>" \
 162: "<Request>" \
 163: "<CommandMsg name= \"FileMove\" id= \"7\">" \
 164: "<VariableRef name= \"FromPathName\"/>" \
 165: "<VariableRef name= \"FromFileName\"/>" \
 166: "<VariableRef name= \"ToPathName\"/>" \
 167: "<VariableRef name= \"ToFileName\"/>" \
 168: "</CommandMsg>" \
 169: "<DataReplyMsg name= \"FileMoveReply\" id= \"8\">" \
 170: "<VariableRef name= \"FromPathName\"/>" \
 171: "<VariableRef name= \"FromFileName\"/>" \
 172: "<VariableRef name= \"ToPathName\"/>" \
 173: "<VariableRef name= \"ToFileName\"/>" \
 174: "<VariableRef name= \"StatusCode\"/>" \
 175: "</DataReplyMsg>" \
 176: "</Request>" \
 177: "<Request>" \
 178: "<CommandMsg name= \"FileDelete\" id= \"9\">" \
 179: "<VariableRef name= \"PathName\"/>" \
 180: "<VariableRef name= \"FileName\"/>" \
 181: "</CommandMsg>" \
 182: "<DataReplyMsg name= \"FileDeleteReply\" id= \"10\">" \
 183: "<VariableRef name= \"PathName\"/>" \
 184: "<VariableRef name= \"FileName\"/>" \
 185: "<VariableRef name= \"StatusCode\"/>" \

```

186: "</DataReplyMsg>" \
187: "</Request>" \
188: "<Request>" \
189: "<CommandMsg name= \"PathCreate \" id= \"11 \">" \
190: "<VariableRef name= \"PathName \"/>" \
191: "</CommandMsg>" \
192: "<DataReplyMsg name= \"PathCreateReply \" id= \"12 \">" \
193: "<VariableRef name= \"PathName \"/>" \
194: "<VariableRef name= \"StatusCode \"/>" \
195: "</DataReplyMsg>" \
196: "</Request>" \
197: "<Request>" \
198: "<CommandMsg name= \"PathDelete \" id= \"13 \">" \
199: "<VariableRef name= \"PathName \"/>" \
200: "</CommandMsg>" \
201: "<DataReplyMsg name= \"PathDeleteReply \" id= \"14 \">" \
202: "<VariableRef name= \"PathName \"/>" \
203: "<VariableRef name= \"StatusCode \"/>" \
204: "</DataReplyMsg>" \
205: "</Request>" \
206: "</Interface>" \
207: "" \
208: "<Interface name= \"BufferInterface \" id= \"3 \">" \
209: "<Variable name= \"BufferName \" format= \"INT08 \" length= \"80 \" kind= \"Buffer_Name_String  
\" description= \"Requested name of buffer \" />" \
210: "<Variable name= \"ElementLength \" format= \"UINT16 \" kind= \"Data_Size_Bytes \"  
description= \"Size of data buffer in Bytes \" />" \
211: "<Variable name= \"MaxNumElements \" format= \"UINT16 \" kind= \"Max_Number_Elements \"  
description= \"Maximum number of elements that log can contain \" />" \
212: "<Variable name= \"RcvSensorID \" format= \"UINT32 \" kind= \"Sensor_ID \" description=  
\"Sensor ID associated with receiving buffer \" />" \
213: "<Variable name= \"RcvIpAddr \" format= \"UINT32 \" kind= \"IP_Address \" description= \"IP  
Address associated with receiving buffer \" />" \
214: "<Variable name= \"RcvPortID \" format= \"UINT16 \" kind= \"Port_ID \" description= \"Port ID  
associated with receiving buffer \" />" \
215: "<Variable name= \"MySensorID \" format= \"UINT32 \" kind= \"Sensor_ID \" description=  
\"Sensor ID of application consuming drain data \" />" \
216: "<Variable name= \"MyIpAddr \" format= \"UINT32 \" kind= \"IP_Address \" description= \"IP  
Address of application consuming drain data \" />" \
217: "<Variable name= \"MyPortID \" format= \"UINT16 \" kind= \"Port_ID \" description= \"Port ID of  
application consuming drain data \" />" \
218: "<Variable name= \"BufferFillLevel \" format= \"FLOAT32 \" kind= \"Fill_Level \" description=  
\"Buffer fill level in percent \" />" \

```

219: "<Variable name= \"NumTotalElements \" format= \"UINT32 \" kind= \"Num_Elements \" description= \"Number of elements stored in buffer \" />\" \

220: "<Variable name= \"NumReadElements \" format= \"UINT32 \" kind= \"Num_Read_Elements \" description= \"Number of elements that have been read but not cleared \" />\" \

221: "<Variable name= \"NumUnreadElements \" format= \"UINT32 \" kind= \"Num_Unread_Elements \" description= \"Number of elements that have not been read \" />\" \

222: "<Variable name= \"SourceSensorID \" format= \"UINT32 \" kind= \"Sensor_ID \" description= \"Sensor ID of qualified search data \" />\" \

223: "<Variable name= \"SourceIpAddr \" format= \"UINT32 \" kind= \"IP_Address \" description= \"IP Address of qualified search data \" />\" \

224: "<Variable name= \"SourcePortID \" format= \"UINT16 \" kind= \"Port_ID \" description= \"Port ID of qualified search data \" />\" \

225: "<Variable name= \"DataSource \" format= \"UINT32 \" kind= \"Component_ID \" description= \"ComponentID of message origination for buffer searches, -1 indicates no constraint \" />\" \

226: "<Variable name= \"MinTime \" format= \"INT32 \" kind= \"Time \" units= \"Seconds \" description= \"Min time range value for buffer searches, -1 indicates no constraint \" />\" \

227: "<Variable name= \"MaxTime \" format= \"INT32 \" kind= \"Time \" units= \"Seconds \" description= \"Max time range value for buffer searches, -1 indicates no constraint \" />\" \

228: "<Variable name= \"SourceSearchFlag \" format= \"UINT08 \" kind= \"Source_Search_Flag \" description= \"Flag indicating search source data, 1 is all, 2 uses Source data from SourceSensorID/SourceIpAddress/SourcePortID variables \" >\" \

229: "<Drange name= \"SearchFlagCodes \">\" \

230: "<Option name= \"AllSources \" value= \"1 \" />\" \

231: "<Option name= \"SelectSource \" value= \"2 \" />\" \

232: "</Drange>\" \

233: "</Variable>\" \

234: "<Variable name= \"ReadMask \" format= \"UINT08 \" kind= \"Read_Mask_Mode \" description= \"Defines whether previously read elements are to be searched \" >\" \

235: "<Drange name= \"ReadMaskCodes \">\" \

236: "<Option name= \"UnreadOnly \" value= \"1 \" />\" \

237: "<Option name= \"ReadOnly \" value= \"2 \" />\" \

238: "<Option name= \"UnreadAndRead \" value= \"3 \" />\" \

239: "</Drange>\" \

240: "</Variable>\" \

241: "<Variable name= \"DrainStyle \" format= \"UINT08 \" kind= \"Drain_Mode \" description= \"Defines whether to drain the buffer Last-In-First-Out or First-In-First-Out \" >\" \

242: "<Drange name= \"DrainStyleCodes \">\" \

243: "<Option name= \"FIFO \" value= \"1 \" />\" \

244: "<Option name= \"LIFO \" value= \"2 \" />\" \

245: "</Drange>\" \

246: "</Variable>\" \

247: "<Variable name= \"BufferInterfaceResponseCode \" format= \"UINT08 \" kind= \"Response_Code \" description= \"Failure Response for Buffer Operations \" >\" \

248: "<Drange name= \"BufferInterfaceResponseCodeTypes \">\" \

249: "<Option name= \"OperationOK \" value= \"1 \"/>" \
 250: "<Option name= \"NameInUse \" value= \"2 \"/>" \
 251: "<Option name= \"InsufficientMemory \" value= \"3 \"/>" \
 252: "<Option name= \"InvalidTimeRange \" value= \"4 \"/>" \
 253: "<Option name= \"NoSuchBuffer \" value= \"5 \"/>" \
 254: "<Option name= \"NoElementAvailable \" value= \"6 \"/>" \
 255: "</Drange>" \
 256: "</Variable>" \
 257: "" \
 258: "<Request>" \
 259: "<CommandMsg name= \"BufferCreate \" id= \"1 \"/>" \
 260: "<VariableRef name= \"BufferName \" />" \
 261: "<VariableRef name= \"ElementLength \" />" \
 262: "<VariableRef name= \"MaxNumElements \" />" \
 263: "</CommandMsg>" \
 264: "<DataReplyMsg name= \"BufferCreateReply \" id= \"2 \"/>" \
 265: "<VariableRef name= \"BufferName \" />" \
 266: "<VariableRef name= \"BufferInterfaceResponseCode \" />" \
 267: "<VariableRef name= \"RcvSensorID \" />" \
 268: "<VariableRef name= \"RcvIpAddr \" />" \
 269: "<VariableRef name= \"RcvPortID \" />" \
 270: "</DataReplyMsg>" \
 271: "</Request>" \
 272: "<Request>" \
 273: "<CommandMsg name= \"BufferSetDrainBehavior \" id= \"3 \"/>" \
 274: "<VariableRef name= \"BufferName \" />" \
 275: "<VariableRef name= \"DataSource \" />" \
 276: "<VariableRef name= \"MinTime \" />" \
 277: "<VariableRef name= \"MaxTime \" />" \
 278: "<VariableRef name= \"ReadMask \" />" \
 279: "</CommandMsg>" \
 280: "<DataReplyMsg name= \"SetDrainFail \" id= \"4 \"/>" \
 281: "<VariableRef name= \"BufferName \" />" \
 282: "<VariableRef name= \"BufferInterfaceResponseCode \" />" \
 283: "</DataReplyMsg>" \
 284: "</Request>" \
 285: "<Request>" \
 286: "<CommandMsg name= \"BufferGetDrainBehavior \" id= \"5 \"/>" \
 287: "<VariableRef name= \"BufferName \" />" \
 288: "</CommandMsg>" \
 289: "<DataReplyMsg name= \"GetDrainResponse \" id= \"6 \"/>" \

290: "<VariableRef name= \"BufferName \" />" \
 291: "<VariableRef name= \"BufferInterfaceResponseCode \" />" \
 292: "<VariableRef name= \"SourceSearchFlag \" />" \
 293: "<VariableRef name= \"SourceSensorID \" />" \
 294: "<VariableRef name= \"SourceIpAddr \" />" \
 295: "<VariableRef name= \"SourcePortID \" />" \
 296: "<VariableRef name= \"MinTime \" />" \
 297: "<VariableRef name= \"MaxTime \" />" \
 298: "<VariableRef name= \"ReadMask \" />" \
 299: "</DataReplyMsg>" \
 300: "</Request>" \
 301: "<Request>" \
 302: "<CommandMsg name= \"BufferGetStatus \" id= \"7 \">" \
 303: "<VariableRef name= \"BufferName \" />" \
 304: "</CommandMsg>" \
 305: "<DataReplyMsg name= \"GetBufferStatusResponse \" id= \"8 \">" \
 306: "<VariableRef name= \"BufferName \" />" \
 307: "<VariableRef name= \"BufferInterfaceResponseCode \" />" \
 308: "<VariableRef name= \"BufferFillLevel \" />" \
 309: "<VariableRef name= \"NumTotalElements \" />" \
 310: "<VariableRef name= \"NumReadElements \" />" \
 311: "<VariableRef name= \"NumUnreadElements \" />" \
 312: "</DataReplyMsg>" \
 313: "</Request>" \
 314: "<Request>" \
 315: "<CommandMsg name= \"BufferDelete \" id= \"9 \">" \
 316: "<VariableRef name= \"BufferName \" />" \
 317: "</CommandMsg>" \
 318: "<DataReplyMsg name= \"BufferDeleteResponse \" id= \"10 \">" \
 319: "<VariableRef name= \"BufferName \" />" \
 320: "<VariableRef name= \"BufferInterfaceResponseCode \" />" \
 321: "</DataReplyMsg>" \
 322: "</Request>" \
 323: "<Request>" \
 324: "<CommandMsg name= \"BufferGetNextElement \" id= \"11 \">" \
 325: "<VariableRef name= \"BufferName \" />" \
 326: "<VariableRef name= \"MySensorID \" />" \
 327: "<VariableRef name= \"MyIpAddr \" />" \
 328: "<VariableRef name= \"MyPortID \" />" \
 329: "<VariableRef name= \"DrainStyle \" />" \
 330: "</CommandMsg>" \

```

331: "<DataReplyMsg name= \"BufferGetNextElementReply \" id= \"12 \">>\" \
332: "<VariableRef name= \"BufferName \" />\" \
333: "<VariableRef name= \"ElementLength \" />\" \
334: "<VariableRef name= \"BufferInterfaceResponseCode \" />\" \
335: "</DataReplyMsg>\" \
336: "</Request>\" \
337: "</Interface>\" \
338: "" \
339: "<Interface name= \"LoggerInterface \" id= \"4 \">>\" \
340: "<Variable name= \"LogName \" format= \"INT08 \" length= \"80 \" kind= \"Logfile_String \"
description= \"Requested name of log \" />\" \
341: "<Variable name= \"MaxEntryLength \" format= \"UINT08 \" kind= \"Entry_Size_Chars \"
description= \"Max size of a log entry in characters \" />\" \
342: "<Variable name= \"MaxEntries \" format= \"UINT32 \" kind= \"Max_Log_Entries \" description=
\"Maximum number of entries \" />\" \
343: "<Variable name= \"LogFillLevel \" format= \"FLOAT32 \" kind= \"Fill_Level \" description=
\"Percent of log filled with entries \" />\" \
344: "<Variable name= \"LogTimeSeconds \" format= \"UINT32 \" kind= \"Timestamp_Seconds \"
description= \"Seconds portion of Log Entry Timestamp \" />\" \
345: "<Variable name= \"LogTimeSubseconds \" format= \"UINT32 \" kind= \"Timestamp_Subseconds
\" description= \"Subseconds portion of Log Entry Timestamp \" />\" \
346: "<Variable name= \"LogEntry \" format= \"INT08 \" length= \"80 \" kind= \"String \" description=
\"Text data to be saved to Log \" />\" \
347: "<Variable name= \"LogInterfaceResponseCode \" format= \"UINT08 \" kind= \"Response_Code \"
description= \"Response for Log Interface \" >\" \
348: "<Drange name= \"LogInterfaceResponseCodeTypes \">>\" \
349: "<Option name= \"OperationOK \" value= \"1 \"/>\" \
350: "<Option name= \"NameInUse \" value= \"2 \"/>\" \
351: "<Option name= \"InsufficientMemory \" value= \"3 \"/>\" \
352: "<Option name= \"NoSuchLog \" value= \"4 \"/>\" \
353: "<Option name= \"LogFull \" value= \"5 \"/>\" \
354: "<Option name= \"EntryTooLong \" value= \"6 \"/>\" \
355: "<Option name= \"LogAlreadyOpen \" value= \"7 \"/>\" \
356: "</Drange>\" \
357: "</Variable>\" \
358: "" \
359: "<Request>\" \
360: "<CommandMsg name= \"LogCreate \" id= \"1 \">>\" \
361: "<VariableRef name= \"LogName \" />\" \
362: "<VariableRef name= \"MaxEntryLength \" />\" \
363: "<VariableRef name= \"MaxEntries \" />\" \
364: "</CommandMsg>\" \
365: "<DataReplyMsg name= \"LogCreateReply \" id= \"2 \">>\" \

```


366: "<VariableRef name= \"LogName \"/>" \
 367: "<VariableRef name= \"LogInterfaceResponseCode \"/>" \
 368: "</DataReplyMsg>" \
 369: "</Request>" \
 370: "<Request>" \
 371: "<CommandMsg name= \"LogStatus \" id= \"3 \">>" \
 372: "<VariableRef name= \"LogName \" />" \
 373: "</CommandMsg>" \
 374: "<DataReplyMsg name= \"LogStatusReply \" id= \"4 \">>" \
 375: "<VariableRef name= \"LogName \"/>" \
 376: "<VariableRef name= \"LogFillLevel \"/>" \
 377: "<VariableRef name= \"LogInterfaceResponseCode \"/>" \
 378: "</DataReplyMsg>" \
 379: "</Request>" \
 380: "<Request>" \
 381: "<CommandMsg name= \"LogWrite \" id= \"5 \">>" \
 382: "<VariableRef name= \"LogName \" />" \
 383: "<VariableRef name= \"LogTimeSeconds \" />" \
 384: "<VariableRef name= \"LogTimeSubseconds \" />" \
 385: "<VariableRef name= \"LogEntry \" />" \
 386: "</CommandMsg>" \
 387: "<DataReplyMsg name= \"LogWriteReply \" id= \"6 \">>" \
 388: "<VariableRef name= \"LogName \"/>" \
 389: "<VariableRef name= \"LogInterfaceResponseCode \"/>" \
 390: "</DataReplyMsg>" \
 391: "</Request>" \
 392: "<Request>" \
 393: "<CommandMsg name= \"LogOpen \" id= \"7 \">>" \
 394: "<VariableRef name= \"LogName \" />" \
 395: "</CommandMsg>" \
 396: "<DataReplyMsg name= \"LogOpenReply \" id= \"8 \">>" \
 397: "<VariableRef name= \"LogName \"/>" \
 398: "<VariableRef name= \"LogInterfaceResponseCode \"/>" \
 399: "</DataReplyMsg>" \
 400: "</Request>" \
 401: "<Request>" \
 402: "<CommandMsg name= \"LogClose \" id= \"9 \">>" \
 403: "<VariableRef name= \"LogName \" />" \
 404: "</CommandMsg>" \
 405: "<DataReplyMsg name= \"LogCloseReply \" id= \"10 \">>" \
 406: "<VariableRef name= \"LogName \"/>" \

```

407: "<VariableRef name= \"LogInterfaceResponseCode \"/>" \
408: "</DataReplyMsg>" \
409: "</Request>" \
410: "<Request>" \
411: "<CommandMsg name= \"LogDelete \" id= \"11 \">" \
412: "<VariableRef name= \"LogName \" />" \
413: "</CommandMsg>" \
414: "<DataReplyMsg name= \"LogDeleteReply \" id= \"12 \">" \
415: "<VariableRef name= \"LogName \"/>" \
416: "<VariableRef name= \"LogInterfaceResponseCode \"/>" \
417: "</DataReplyMsg>" \
418: "</Request>" \
419: "" \
420: "</Interface>" \
421: "" \
422: "</xTEDS>" \
423: ""
424:
425: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/TelemetryHandler.h

```
1: #ifndef _TELEMETRYHANDLER_XTEDS_H
2: #define _TELEMETRYHANDLER_XTEDS_H
3:
4: #define _STRING_TELEMETRYHANDLER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"TelemetryHandlerXTEDS \" description= \"TelemetryHandler xTEDS \" version= \"1.0\">\" \
8:   \"\" \
9:   \"<Application name= \"TelemetryHandler \" kind= \"CommunicationsFlightSoftware \" description=
\"Communications Subsystem Telemetry Handler \"/>\" \
10:   \"\" \
11:   \"<Interface name= \"TelemetryHandlerInterface \" id= \"1 \" description= \"Basic Telemetry Manager
interface \"/>\" \
12:   \"\" \
13:   \"<Variable name= \"ComponentSensorID \"      kind= \"tbd \"      format= \"UINT32
\"/>\" \
14:   \"<Variable name= \"MsgID \"      kind= \"tbd \" rangeMin= \"1 \" rangeMax= \"255 \"
format= \"UINT08 \"/>\" \
15:   \"<Variable name= \"InterfaceID \"      kind= \"tbd \" rangeMin= \"0 \" rangeMax= \"255 \"
format= \"UINT08 \"/>\" \
16:   \"<Variable name= \"ComponentTelemetryLevel \" kind= \"tbd \" rangeMin= \"0 \" rangeMax= \"4 \"
format= \"UINT08 \"/>\" \
17:   \"<Variable name= \"MsgTelemetryLevel \"      kind= \"tbd \" rangeMin= \"1 \" rangeMax= \"3 \"
format= \"UINT08 \"/>\" \
18:   \"<Variable name= \"SendX \"      kind= \"tbd \"      format= \"UINT08 \"/>\" \
19:   \"<Variable name= \"OfY \"      kind= \"tbd \"      format= \"UINT08 \"/>\" \
20:   \"\" \
21:   \"<Command>\" \
22:   \"<CommandMsg name= \"RestoreMsgTlmLevel \" id= \"002 \" description= \"Restore the telemetry
level of the message to default level \"/>\" \
23:   \"<VariableRef name= \"ComponentSensorID \"/>\" \
24:   \"<VariableRef name= \"MsgID \"/>\" \
25:   \"<VariableRef name= \"InterfaceID \"/>\" \
26:   \"</CommandMsg>\" \
27:   \"</Command>\" \
28:   \"\" \
29:   \"<Command>\" \
30:   \"<CommandMsg name= \"SetComponentTlmLevel \" id= \"003 \" description= \"Set the telemetry
level of this component \"/>\" \
```

31: "<VariableRef name= \"ComponentSensorID \"/>\" \\
 32: "<VariableRef name= \"ComponentTelemetryLevel \"/>\" \\
 33: "</CommandMsg>\" \\
 34: "</Command>\" \\
 35: "" \\
 36: "<Command>\" \\
 37: "<CommandMsg name= \"SetComponentTlmLevelToDefault \" id= \"004 \" description= \"Set the telemetry level of this component to default \">>\" \\
 38: "<VariableRef name= \"ComponentSensorID \"/>\" \\
 39: "</CommandMsg>\" \\
 40: "</Command>\" \\
 41: "" \\
 42: "<Command>\" \\
 43: "<CommandMsg name= \"SetDefaultTlmLevel \" id= \"005 \" description= \"Set the overall telemetry level of the SYSTEM \">>\" \\
 44: "<VariableRef name= \"ComponentTelemetryLevel \"/>\" \\
 45: "</CommandMsg>\" \\
 46: "</Command>\" \\
 47: "" \\
 48: "<Command>\" \\
 49: "<CommandMsg name= \"SetMsgTlmLevel \" id= \"006 \" description= \"Set the telemetry level of the message \">>\" \\
 50: "<VariableRef name= \"ComponentSensorID \"/>\" \\
 51: "<VariableRef name= \"MsgID \"/>\" \\
 52: "<VariableRef name= \"InterfaceID \"/>\" \\
 53: "<VariableRef name= \"MsgTelemetryLevel \"/>\" \\
 54: "</CommandMsg>\" \\
 55: "</Command>\" \\
 56: "" \\
 57: "<Command>\" \\
 58: "<CommandMsg name= \"SetMsgDownlinkRate \" id= \"007 \" description= \"Set the rate at which this message will be downlinked \">>\" \\
 59: "<VariableRef name= \"ComponentSensorID \"/>\" \\
 60: "<VariableRef name= \"MsgID \"/>\" \\
 61: "<VariableRef name= \"InterfaceID \"/>\" \\
 62: "<VariableRef name= \"SendX \"/>\" \\
 63: "<VariableRef name= \"OfY \"/>\" \\
 64: "</CommandMsg>\" \\
 65: "</Command>\" \\
 66: "" \\
 67: "</Interface>\" \\
 68: "" \

```

69: "<Interface name= \"TelemetryHandlerStatusInterface \" id= \"2 \">" \
70: "" \
71: "<Variable name= \"TelemetryHandlerStatus \" kind= \"Status \" format= \"UINT08 \">" \
72: "<Drange name= \"TelemetryHandlerStatusEnum \">" \
73: "<Option value= \"0 \" name= \"NOT_INITIALIZED \"/><!-- The TelemetryHandler has not been
successfully initialized -->" \
74: "<Option value= \"1 \" name= \"INITIALIZING \"/>" \
75: "<Option value= \"2 \" name= \"RUNNING \"/><!-- The TelemetryHandler is initialized and is
running -->" \
76: "<Option value= \"3 \" name= \"TERMINATING \"/><!-- The TelemetryHandler is shutting down --
>" \
77: "</Drange>" \
78: "</Variable>" \
79: "" \
80: "<Variable name= \"SystemTelemetryLevel \" kind= \"tbd \" rangeMin= \"0 \" rangeMax= \"4 \"
format= \"UINT08 \">" \
81: "<Variable name= \"DataMsgsReceived \" kind= \"tbd \" format= \"UINT32 \"/>" \
82: "<Variable name= \"DataMsgsForwarded \" kind= \"tbd \" format= \"UINT32 \"/>" \
83: "<Variable name= \"ComponentsRegistered \" kind= \"tbd \" format= \"UINT16 \"/>" \
84: "<Variable name= \"DataMsgsRegistered \" kind= \"tbd \" format= \"UINT16 \"/>" \
85: "<Variable name= \"DataMsgsSubscribed \" kind= \"tbd \" format= \"UINT16 \"/>" \
86: "<Variable name= \"CommandsReceived \" kind= \"tbd \" format= \"UINT32 \"/>" \
87: "<Variable name= \"CommandsAccepted \" kind= \"tbd \" format= \"UINT32 \"/>" \
88: "<Variable name= \"CommandsRejected \" kind= \"tbd \" format= \"UINT32 \"/>" \
89: "" \
90: "<Notification>" \
91: "<DataMsg name= \"TelemetryHandlerStatusMsg \" id= \"001 \" msgArrival= \"EVENT \">" \
92: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
93: "<VariableRef name= \"TelemetryHandlerStatus \"/>" \
94: "<VariableRef name= \"ComponentsRegistered \"/>" \
95: "<VariableRef name= \"DataMsgsReceived \"/>" \
96: "<VariableRef name= \"DataMsgsForwarded \"/>" \
97: "<VariableRef name= \"DataMsgsRegistered \"/>" \
98: "<VariableRef name= \"DataMsgsSubscribed \"/>" \
99: "<VariableRef name= \"CommandsReceived \"/>" \

```

```

100: "<VariableRef name= \"CommandsAccepted \"/>" \
101: "<VariableRef name= \"CommandsRejected \"/>" \
102: "<VariableRef name= \"SystemTelemetryLevel \"/>" \
103: "</DataMsg>" \
104: "</Notification>" \
105: "" \
106: "</Interface>" \
107: "" \
108: "<Interface name= \"ICSDebugInterface \" id= \"3 \"/>" \
109: "" \
110: "<!--" \
111: "Note: DebugLevel is a bit field with the following assigned values:" \
112: "DEBUG_ENTRY_AND_EXIT = 0x01," \
113: "DEBUG_ENTRY_PARAMETERS = 0x02," \
114: "DEBUG_EXIT_PARAMETERS = 0x04," \
115: "DEBUG_LEVEL_LOW = 0x08," \
116: "DEBUG_LEVEL_MEDIUM = 0x10," \
117: "DEBUG_LEVEL_HIGH = 0x20." \
118: "The values are OR'd to determine the debug information to be logged." \
119: "-->" \
120: "" \
121: "<Variable name= \"DebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
122: "<Variable name= \"CurrentDebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
123: "" \
124: "<Variable name= \"SetDebugLevelReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \
125: "<Variable name= \"SetDebugLevelAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \
126: "<Variable name= \"SetDebugLevelSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \
127: "<Variable name= \"SetDebugLevelFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \
128: "" \
129: "<Command>" \
130: "<CommandMsg name= \"SetDebugLevel \" id= \"001 \" description= \"Set the debug log verbosity level \"/>" \
131: "<VariableRef name= \"DebugLevel \"/>" \
132: "</CommandMsg>" \
133: "</Command>" \
134: "" \
135: "<Command>" \
136: "<CommandMsg name= \"SendDebugStatus \" id= \"002 \"/>" \
137: "</Command>" \
138: "" \
139: "<Notification>" \

```

```

140: "<DataMsg name= \"DebugStatus \" id= \"03 \" msgArrival= \"EVENT \">" \
141: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
142: "<VariableRef name= \"CurrentDebugLevel \"/>" \
143: "<VariableRef name= \"SetDebugLevelReceived \"/>" \
144: "<VariableRef name= \"SetDebugLevelAccepted \"/>" \
145: "<VariableRef name= \"SetDebugLevelSuccess \"/>" \
146: "<VariableRef name= \"SetDebugLevelFailure \"/>" \
147: "</DataMsg>" \
148: "</Notification>" \
149: "" \
150: "</Interface>" \
151: "" \
152: "<Interface name= \"TaskControlInterface \" id= \"4 \">" \
153: "" \
154: "<Command>" \
155: "<CommandMsg name= \"DestroyTask \" id= \"001 \"/>" \
156: "</Command>" \
157: "" \
158: "<!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->"
\
159: "<!-- Other possible requests include GetPriority, and GetHeartBeatCount -->" \
160: "" \
161: "</Interface>" \
162: "" \
163: "" \
164: "</xTEDS>" \
165: ""
166:
167: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/DigitalSS.h

```
1: #ifndef _ADCOLENRLDIGITALSUNSENSOR_XTEDS_H
2: #define _ADCOLENRLDIGITALSUNSENSOR_XTEDS_H
3:
4: #define _STRING_ADCOLENRLDIGITALSUNSENSOR_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
7:   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
8:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
9:   name= \"AdcoleNRLDigitalSunSensorxTeds \"\" \
10:   \"version= \"2.0 \">\" \
11:   \"<Device name= \"AdcoleNRLDigitalSunSensor \" kind= \"fss \" description= \"A single 2-Axis
12:   Digital Sun Sensor \" />\" \
13:   \"<Interface name= \"DigitalSunSensorInterface \" id= \"1 \">\" \
14:   \"<Qualifier name= \"headID \" value= \"0 \" />\" \
15:   \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
16:   \"<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
17:   scaleFactor= \".0001\" scaleUnits= \"Seconds \" />\" \
18:   \"<Variable name= \"Sun_Presence \" kind= \"Valid \" format= \"UINT08 \">\" \
19:   \"<Drange name= \"Sun_PresenceEnum \">\" \
20:   \"<Option name= \"SunPresent \" value= \"1 \" description= \"This sensor can see the sun \" />\" \
21:   \"<Option name= \"SunNotPresent \" value= \"0 \" description= \"Sun not visible from this sensor \"
22:   />\" \
23:   \"</Drange>\" \
24:   \"</Variable>\" \
25:   \"<Variable name= \"SunAngle \" kind= \"SunAngle \" units= \"deg \" format= \"FLOAT32 \" length=
26:   \"2 \" rangeMin= \"-128 \"\" \
27:   \"rangeMax= \"128 \" accuracy= \"0.25 \">\" \
28:   \"<Qualifier name= \"Frame_Measured \" value= \"DVF \" />\" \
29:   \"</Variable>\" \
30:   \"<Notification>\" \
31:   \"<DataMsg id= \"1 \" name= \"AngleToSun \" msgArrival= \"PERIODIC \" msgRate= \"100 \">\" \
32:   \"<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
33:   \"<VariableRef name= \"Time \" />\" \
34:   \"<VariableRef name= \"SubS \" />\" \
35:   \"<VariableRef name= \"Sun_Presence \" />\" \
36:   \"<VariableRef name= \"SunAngle \" />\" \
37:   \"</DataMsg>\" \
38:   \"</Notification>\" \
39:   \"</Interface>\" \
```



```

35: "" \
36: "<Interface name= \"DevPwr\" id= \"2\">\" \
37: "<Qualifier name= \"CurrentLoKeepout\" value= \"0.0\" units= \"A\"/>\" \
38: "<Qualifier name= \"CurrentLoWarning\" value= \"0.0\" units= \"A\"/>\" \
39: "<Qualifier name= \"CurrentHiWarning\" value= \"3.5\" units= \"A\"/>\" \
40: "<Qualifier name= \"CurrentHiKeepout\" value= \"3.5\" units= \"A\"/>\" \
41: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />\" \
42: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\" \
scaleFactor= \".0001\" scaleUnits= \"Seconds\" />\" \
43: "<Variable name= \"DevPwrState\" kind= \"Power_State\" format= \"UINT08\">\" \
44: "<Drange name= \"DevPwrStateEnum\">\" \
45: "<Option name= \"DevPwrOFF\" value= \"0\" description= \"All power to device is turned off.\" />\" \
46: "<Option name= \"DevPwrON\" value= \"1\" description= \"Device may draw full power.\" />\" \
47: "</Drange>\" \
48: "</Variable>\" \
49: "<Variable name= \"DevPwrStateSet\" kind= \"Power_State\" format= \"UINT08\" id= \"2\">\" \
50: "<Drange name= \"DevPwrStateEnumReference\" description= \"This should be a reference to \
DevPwrStateEnumeration.\">\" \
51: "<Option name= \"DevPwrOFF\" value= \"0\" description= \"All power to device is turned off.\" />\" \
52: "<Option name= \"DevPwrON\" value= \"1\" description= \"Device may draw full power.\" />\" \
53: "</Drange>\" \
54: "</Variable>\" \
55: "<Variable name= \"modePowers\" kind= \"power\" format= \"FLOAT32\" units= \"W\" length= \
\"2\" />\" \
56: "<Command>\" \
57: "<CommandMsg name= \"DevPwrSetState\" id= \"1\">\" \
58: "<VariableRef name= \"DevPwrStateSet\" />\" \
59: "</CommandMsg>\" \
60: "<FaultMsg name= \"DevPwrStateNotSet\" id= \"2\">\" \
61: "<VariableRef name= \"Time\" />\" \
62: "<VariableRef name= \"SubS\" />\" \
63: "<VariableRef name= \"DevPwrState\" />\" \
64: "<VariableRef name= \"DevPwrStateSet\" />\" \
65: "</FaultMsg>\" \
66: "</Command>\" \
67: "<Notification>\" \
68: "<DataMsg name= \"DevPwrHK\" id= \"3\" msgArrival= \"PERIODIC\">\" \
69: "<Qualifier name= \"telemetryLevel\" value= \"1\"/>\" \
70: "<VariableRef name= \"Time\" />\" \
71: "<VariableRef name= \"SubS\" />\" \

```

```

72: "<VariableRef name= \"DevPwrState \" />" \
73: "<VariableRef name= \"DevPwrStateSet \" />" \
74: "</DataMsg>" \
75: "</Notification>" \
76: "<Request>" \
77: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
78: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">" \
79: "<VariableRef name= \"modePowers \" />" \
80: "</DataReplyMsg>" \
81: "</Request>" \
82: "</Interface>" \
83: "" \
84: "<Interface name= \"CmpSOH \" id= \"3 \">" \
85: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \" />" \
86: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \" />" \
87: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \" />" \
88: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \" />" \
89: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
90: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
91: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \" />" \
92: "<Request>" \
93: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />" \
94: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">" \
95: "<VariableRef name= \"Time \" />" \
96: "<VariableRef name= \"SubS \" />" \
97: "<VariableRef name= \"DeviceTemperature \" />" \
98: "</DataReplyMsg>" \
99: "</Request>" \
100: "<Notification>" \
101: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
102: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
103: "<VariableRef name= \"Time \" />" \
104: "<VariableRef name= \"SubS \" />" \
105: "<VariableRef name= \"DeviceTemperature \" />" \
106: "</DataMsg>" \
107: "</Notification>" \
108: "</Interface>" \
109: "</xTEDS>" \
110: ""

```

111:
112: #endif

File: sdm/app/test/DMTests/xTEDSRegTests/ChargeBatteries.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS    xTEDS02.xsd"
name="ActivityAgentXTEDS" description="ActivityAgent xTEDS" version="2.3">
4:
5:     <Application name="ChargeBatteries" kind="AutonomyFlightSoftware" description="Autonomous
Tasking Executive (ATE), ActivityAgent"/>
6:
7:     <!-- Note: An ActivityAgent must implement the ActivityInterface and should implement the
ActivityAgentStatusInterface. -->
8:
9:     <Interface name="ActivityInterface" id="1"><!-- This is a template for an ActivityAgent Interface.
Unique activity parameters must be added as necessary. -->
10:
11:         <Variable name="ActivityName" kind="tbd" format="UINT08" length="33"/><!-- 33-byte, null
terminated string -->
12:         <Variable name="ActivityId" kind="ID" format="UINT32"/>
13:         <Variable name="ActivityStatus" kind="Status" format="INT16">
14:             <Drange name="ActivityStatusEnum">
15:                 <Option value="0" name="SCHEDULE_FAILURE"/><!-- The activity could not be scheduled
as requested -->
16:                 <Option value="1" name="WAITING"/><!-- The activity has been inserted into the schedule -->
17:                 <Option value="2" name="ENABLED"/><!-- The activity has been sent a command to execute -
->
18:                 <Option value="3" name="TERMINATED"/><!-- The activity has been sent a command to
abort -->
19:                 <Option value="4" name="EXECUTING"/><!-- The activity is currently executing -->
20:                 <Option value="5" name="DONE_FAILURE"/><!-- The activity has completed abnormally -->
21:                 <Option value="6" name="DONE_SUCCESS"/><!-- The activity has completed normally -->
22:                 <Option value="7" name="DONE_NOT_EXECUTED"/><!-- The activity has terminated
without executing -->
23:             </Drange>
24:         </Variable>
25:
26:     <!-- Add Variables for the unique activity parameters -->
27:
28:     <Command> <!-- Provides the capability to request an Activity from the ground -->
29:         <CommandMsg name="RequestActivityCmd" id="001" description="Determine activity state
variables and request to be scheduled">
```

```

30:     <VariableRef name="ActivityId"/><!-- Use 0 for on-board requests. The ActivityId will then be
assigned by the ActivityManager -->
31:
32:     <!-- Add VariableRefs for the unique activity parameters -->
33:
34: </CommandMsg>
35: </Command>
36:
37: <Request>
38:     <CommandMsg name="RequestActivity" id="002" description="Determine activity state
variables and request to be scheduled">
39:     <VariableRef name="ActivityId"/><!-- Use 0 for on-board requests. The ActivityId will then be
assigned by the ActivityManager -->
40:
41:     <!-- Add VariableRefs for the unique activity parameters -->
42:
43: </CommandMsg>
44: <DataReplyMsg name="RequestActivityReply" id="003">
45:     <VariableRef name="ActivityId"/>
46:     <VariableRef name="ActivityStatus"/>
47: </DataReplyMsg>
48: </Request>
49:
50: <Command> <!-- Provides the capability to update an Activity from the ground -->
51:     <CommandMsg name="UpdateRequestCmd" id="004" description="Update activity state
variables and request a schedule update if necessary">
52:     <VariableRef name="ActivityId"/>
53:
54:     <!-- Add VariableRefs for the unique activity parameters -->
55:
56: </CommandMsg>
57: </Command>
58:
59: <Request>
60:     <CommandMsg name="UpdateRequest" id="005" description="Update activity state variables
and request a schedule update if necessary">
61:     <VariableRef name="ActivityId"/>
62:
63:     <!-- Add VariableRefs for the unique activity parameters -->
64:
65: </CommandMsg>
66: <DataReplyMsg name="UpdateRequestReply" id="006">

```

```

67:     <VariableRef name="ActivityId"/>
68:     <VariableRef name="ActivityStatus"/>
69: </DataReplyMsg>
70: </Request>
71:
72: <Command>
73:     <CommandMsg name="Reschedule" id="007" description="The activity has been removed from
the schedule and needs to be rescheduled">
74:         <VariableRef name="ActivityId"/>
75:     </CommandMsg>
76: </Command>
77:
78: <Command>
79:     <CommandMsg name="Delete" id="008" description="Delete the activity">
80:         <VariableRef name="ActivityId"/>
81:     </CommandMsg>
82: </Command>
83:
84: <Command>
85:     <CommandMsg name="Execute" id="009" description="Execute the activity">
86:         <VariableRef name="ActivityId"/>
87:     </CommandMsg>
88: </Command>
89:
90: <Command>
91:     <CommandMsg name="Abort" id="010" description="Abort execution of the activity">
92:         <VariableRef name="ActivityId"/>
93:     </CommandMsg>
94: </Command>
95:
96: <Command>
97:     <CommandMsg name="SendActivityStatusMsg" id="011" description="Send the
ActivityStatusMsg DataMsg">
98:         <VariableRef name="ActivityId"/>
99:     </CommandMsg>
100: </Command>
101:
102: <Notification>
103:     <DataMsg name="ActivityStatusMsg" id="012" msgArrival="EVENT">
104:         <Qualifier value="1" name="telemetryLevel"/>
105:         <VariableRef name="ActivityId"/>

```

```

106:     <VariableRef name="ActivityName"/>
107:     <VariableRef name="ActivityStatus"/>
108: </DataMsg>
109: </Notification>
110:
111: </Interface>
112:
113: <Interface name="ActivityAgentStatusInterface" id="2">
114:
115:     <Variable name="ActivityAgentStatus" kind="Status" format="INT16">
116:         <Drange name="ActivityAgentStatusEnum">
117:             <Option value="0" name="NOT_INITIALIZED"/><!-- The ActivityAgent has not been
successfully initialized -->
118:             <Option value="1" name="INITIALIZING"/><!-- The ActivityAgent is in the process of
initializing -->
119:             <Option value="2" name="RUNNING"/><!-- The ActivityAgent is initialized and is running --
>
120:             <Option value="3" name="TERMINATING"/><!-- The ActivityAgent is shutting down -->
121:         </Drange>
122:     </Variable>
123:
124:     <Variable name="ActivityAgentName" kind="tbd" format="UINT08" length="33"/><!-- 33-byte,
null terminated string -->
125:
126:     <Variable name="ActivitiesCurrentlyScheduled" kind="tbd" format="UINT16"/>
127:
128:     <Variable name="ActivitiesExecuted" kind="tbd" format="UINT16"/>
129:     <Variable name="ActivitiesExecutedSuccess" kind="tbd" format="UINT16"/>
130:     <Variable name="ActivitiesExecutedFailure" kind="tbd" format="UINT16"/>
131:     <Variable name="ActivitiesDeleted" kind="tbd" format="UINT16"/>
132:
133:     <Variable name="RequestActivityReceived" kind="tbd" format="UINT16"/>
134:     <Variable name="RequestActivityAccepted" kind="tbd" format="UINT16"/>
135:     <Variable name="RequestActivitySuccess" kind="tbd" format="UINT16"/>
136:     <Variable name="RequestActivityFailure" kind="tbd" format="UINT16"/>
137:
138:     <Variable name="UpdateRequestReceived" kind="tbd" format="UINT16"/>
139:     <Variable name="UpdateRequestAccepted" kind="tbd" format="UINT16"/>
140:     <Variable name="UpdateRequestSuccess" kind="tbd" format="UINT16"/>
141:     <Variable name="UpdateRequestFailure" kind="tbd" format="UINT16"/>
142:
143:     <Variable name="ExecuteReceived" kind="tbd" format="UINT16"/>

```

```

144: <Variable name="ExecuteAccepted" kind="tbd" format="UINT16"/>
145: <Variable name="ExecuteSuccess" kind="tbd" format="UINT16"/>
146: <Variable name="ExecuteFailure" kind="tbd" format="UINT16"/>
147:
148: <Variable name="RescheduleReceived" kind="tbd" format="UINT16"/>
149: <Variable name="RescheduleAccepted" kind="tbd" format="UINT16"/>
150: <Variable name="RescheduleSuccess" kind="tbd" format="UINT16"/>
151: <Variable name="RescheduleFailure" kind="tbd" format="UINT16"/>
152:
153: <Variable name="DeleteReceived" kind="tbd" format="UINT16"/>
154: <Variable name="DeleteAccepted" kind="tbd" format="UINT16"/>
155: <Variable name="DeleteSuccess" kind="tbd" format="UINT16"/>
156: <Variable name="DeleteFailure" kind="tbd" format="UINT16"/>
157:
158: <Command>
159:     <CommandMsg name="SendActivityAgentStatusMsg" id="001" description="Send the
ActivityAgentStatusMsg DataMsg"/>
160: </Command>
161:
162: <Notification>
163:     <DataMsg name="ActivityAgentStatusMsg" id="002" msgArrival="EVENT">
164:         <Qualifier value="1" name="telemetryLevel"/>
165:         <VariableRef name="ActivityAgentStatus"/>
166:         <VariableRef name="ActivityAgentName"/>
167:         <VariableRef name="ActivitiesCurrentlyScheduled"/>
168:         <VariableRef name="ActivitiesExecuted"/>
169:         <VariableRef name="ActivitiesExecutedSuccess"/>
170:         <VariableRef name="ActivitiesExecutedFailure"/>
171:         <VariableRef name="ActivitiesDeleted"/>
172:         <VariableRef name="RequestActivityReceived"/>
173:         <VariableRef name="RequestActivityAccepted"/>
174:         <VariableRef name="RequestActivitySuccess"/>
175:         <VariableRef name="RequestActivityFailure"/>
176:         <VariableRef name="UpdateRequestReceived"/>
177:         <VariableRef name="UpdateRequestAccepted"/>
178:         <VariableRef name="UpdateRequestSuccess"/>
179:         <VariableRef name="UpdateRequestFailure"/>
180:         <VariableRef name="ExecuteReceived"/>
181:         <VariableRef name="ExecuteAccepted"/>
182:         <VariableRef name="ExecuteSuccess"/>
183:         <VariableRef name="ExecuteFailure"/>

```



```

184:     <VariableRef name="RescheduleReceived"/>
185:     <VariableRef name="RescheduleAccepted"/>
186:     <VariableRef name="RescheduleSuccess"/>
187:     <VariableRef name="RescheduleFailure"/>
188:     <VariableRef name="DeleteReceived"/>
189:     <VariableRef name="DeleteAccepted"/>
190:     <VariableRef name="DeleteSuccess"/>
191:     <VariableRef name="DeleteFailure"/>
192: </DataMsg>
193: </Notification>
194:
195: </Interface>
196:
197: <Interface name="ATEDebugInterface" id="3">
198:
199:     <!--
200:     Note: DebugLevel is a bit field with the following assigned values:
201:         DEBUG_ENTRY_AND_EXIT   = 0x01,
202:         DEBUG_ENTRY_PARAMETERS = 0x02,
203:         DEBUG_EXIT_PARAMETERS  = 0x04,
204:         DEBUG_LEVEL_LOW        = 0x08,
205:         DEBUG_LEVEL_MEDIUM     = 0x10,
206:         DEBUG_LEVEL_HIGH       = 0x20.
207:     The values are OR'd to determine the debug information to be logged.
208:     -->
209:
210:     <Variable name="DebugLevel" kind="tbd" format="UINT16"/>
211:     <Variable name="CurrentDebugLevel" kind="tbd" format="UINT16"/>
212:
213:     <Variable name="SetDebugLevelReceived" kind="tbd" format="UINT16"/>
214:     <Variable name="SetDebugLevelAccepted" kind="tbd" format="UINT16"/>
215:     <Variable name="SetDebugLevelSuccess" kind="tbd" format="UINT16"/>
216:     <Variable name="SetDebugLevelFailure" kind="tbd" format="UINT16"/>
217:
218:     <Command>
219:         <CommandMsg name="SetDebugLevel" id="001" description="Set the debug log verbosity
level">
220:             <VariableRef name="DebugLevel"/>
221:         </CommandMsg>
222:     </Command>
223:

```

```

224: <Command>
225:   <CommandMsg name="SendDebugStatus" id="002"/>
226: </Command>
227:
228: <Notification>
229:   <DataMsg name="DebugStatus" id="003" msgArrival="EVENT">
230:     <Qualifier value="1" name="telemetryLevel"/>
231:     <VariableRef name="CurrentDebugLevel"/>
232:     <VariableRef name="SetDebugLevelReceived"/>
233:     <VariableRef name="SetDebugLevelAccepted"/>
234:     <VariableRef name="SetDebugLevelSuccess"/>
235:     <VariableRef name="SetDebugLevelFailure"/>
236:   </DataMsg>
237: </Notification>
238:
239: </Interface>
240:
241: <Interface name="TaskControlInterface" id="4">
242:
243:   <Command>
244:     <CommandMsg name="DestroyTask" id="001"/>
245:   </Command>
246:
247:   <!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->
248:   <!-- Other possible requests include GetPriority, and GetHeartBeatCount -->
249:
250: </Interface>
251:
252: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/AdcoleDigitalSS.h

```
1: #ifndef _ADCOLE_NRL_DIGITAL_SUN_SENSOR_XTEDS_H
2: #define _ADCOLE_NRL_DIGITAL_SUN_SENSOR_XTEDS_H
3:
4: #define _STRING_ADCOLE_NRL_DIGITAL_SUN_SENSOR_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"Adcole_Digital_Sun_Sensor \" version= \"1.0 \"><!-- plc changed xTEDS schema location -->\"
\
8:   "<Device name= \"Adcole_NRL_Digital_Sun_Sensor \" kind= \"Sun_Sensor \" modelId= \"DSS \"
description= \"Adcole Digital 2-Axis Sun Sensor \"/>\" \
9:   "" \
10:  "<Interface name= \"Sun_Sensor_Interface \" id= \"1 \"><!-- plc added Interface element -->\" \
11:  "" \
12:  "<!-- Timestamp -->\" \
13:  "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"><!-- plc
changed FORMAT value -->\" \
14:  "</Variable>\" \
15:  "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"><!-- plc changed FORMAT value -->\" \
16:  "</Variable>\" \
17:  "" \
18:  "" \
19:  "<!-- Data Definitions -->\" \
20:  "<Variable name= \"Data_Rate \" id= \"3 \" kind= \"Msg_Rate \" rangeMax= \"10000 \" rangeMin=
\"0 \" units= \"Hz \" format= \"INT16 \"/><!-- plc changed FORMAT value -->\" \
21:  "<Variable name= \"ID \" id= \"4 \" kind= \"ID \" format= \"UINT08 \"/><!-- plc changed FORMAT
value -->\" \
22:  "" \
23:  "<Variable name= \"Sensor_Number \" id= \"5 \" kind= \"TBD \" units= \"Counts \" format=
\"UINT16 \"/><!-- plc changed FORMAT value -->\" \
24:  "<Variable name= \"Sun_Presence \" id= \"6 \" kind= \"TBD \" units= \"Counts \" format= \"UINT16
\" rangeMin= \"0 \" rangeMax= \"1 \"><!-- plc changed FORMAT value -->\" \
25:  "<Variable name= \"Sun_Angle_X \" id= \"7 \" kind= \"Angle \" units= \"Degrees \" format=
\"FLOAT32 \" rangeMin= \"-128 \" rangeMax= \"128 \" accuracy= \"0.25 \" /><!-- plc changed
FORMAT value -->\" \
26:  "<Variable name= \"Sun_Angle_Y \" id= \"8 \" kind= \"Angle \" units= \"Degrees \" format=
\"FLOAT32 \" rangeMin= \"-128 \" rangeMax= \"128 \" accuracy= \"0.25 \" /><!-- plc changed
FORMAT value -->\" \
27:  "" \
28:  "<!-- Data Messages -->\" \
```

```

29: "" \
30: "<Notification><!-- plc added Notification element -->" \
31: "<DataMsg id= \"2\" name= \"Sun_Angle\" msgArrival= \"PERIODIC\" msgRate= \"100\">" \
32: "<VariableRef name= \"SubS\"/>" \
33: "<VariableRef name= \"Time\"/>" \
34: "<VariableRef name= \"Sensor_Number\"/>" \
35: "<VariableRef name= \"Sun_Presence\"/>" \
36: "<VariableRef name= \"Sun_Angle_X\"/>" \
37: "<VariableRef name= \"Sun_Angle_Y\"/>" \
38: "</DataMsg>" \
39: "</Notification>" \
40: "" \
41: "<!-- Command Messages -->" \
42: "<Command><!-- plc added Command element -->" \
43: "<CommandMsg id= \"10\" name= \"Power_Off\"/>" \
44: "</Command>" \
45: "" \
46: "<Command><!-- plc added Command element -->" \
47: "<CommandMsg id= \"11\" name= \"Power_On\"/>" \
48: "</Command>" \
49: "" \
50: "<Command><!-- plc added Command element -->" \
51: "<CommandMsg id= \"12\" name= \"Message_Rate\">" \
52: "<VariableRef name= \"ID\"/>" \
53: "<VariableRef name= \"Data_Rate\"/>" \
54: "</CommandMsg>" \
55: "</Command>" \
56: "" \
57: "</Interface>" \
58: "" \
59: "<Interface name= \"DevPwr\" id= \"2\">" \
60: "<Qualifier name= \"CurrentLoKeepout\" value= \"0.0\" units= \"A\"/>" \
61: "<Qualifier name= \"CurrentLoWarning\" value= \"0.0\" units= \"A\"/>" \
62: "<Qualifier name= \"CurrentHiWarning\" value= \"3.5\" units= \"A\"/>" \
63: "<Qualifier name= \"CurrentHiKeepout\" value= \"3.5\" units= \"A\"/>" \
64: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\"/>" \
65: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\" scaleFactor= \".0001\" scaleUnits= \"Seconds\"/>" \
66: "<Variable name= \"DevPwrState\" kind= \"Power_State\" format= \"UINT08\">" \
67: "<Drange name= \"DevPwrStateEnum\">" \
68: "<Option name= \"DevPwrOFF\" value= \"0\"/>" \

```

```

69: "<Option name= \"DevPwrON \" value= \"1 \" />" \
70: "</Drange>" \
71: "</Variable>" \
72: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \">>" \
73: "<Drange name= \"DevPwrStateEnumReference \">" \
74: "<Option name= \"DevPwrOFF \" value= \"0 \" />" \
75: "<Option name= \"DevPwrON \" value= \"1 \" />" \
76: "</Drange>" \
77: "</Variable>" \
78: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length= \"2 \">>" \
79: "<Command>" \
80: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \">>" \
81: "<VariableRef name= \"DevPwrStateSet \" />" \
82: "</CommandMsg>" \
83: "</Command>" \
84: "<Notification>" \
85: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">>" \
86: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
87: "<VariableRef name= \"Time \" />" \
88: "<VariableRef name= \"SubS \" />" \
89: "<VariableRef name= \"DevPwrState \" />" \
90: "<VariableRef name= \"DevPwrStateSet \" />" \
91: "</DataMsg>" \
92: "</Notification>" \
93: "<Request>" \
94: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
95: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">>" \
96: "<VariableRef name= \"modePowers \"/>" \
97: "</DataReplyMsg>" \
98: "</Request>" \
99: "</Interface>" \
100: "</xTEDS>" \
101: ""
102:
103: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/Battery.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="Battery_XTEDS"
4:   version="2.0">
5:   <Device name="Battery" kind="bat" description="power storage device" >
6:     <Qualifier name="Manufacturer" value="DNetConceptBattery"/>
7:     <Qualifier name="Model" value="1.2.3"/>
8:     <Qualifier name="SerialNumber" value="90210"/>
9:   </Device>
10:
11:   <Interface name="BatteryBasicInterface" id="1">
12:     <Qualifier name="Chemistry" value="LiIon"/>
13:     <Qualifier name="Capacity" value="30.0" units="Amp-Hours"/>
14:     <Qualifier name="DepthOfDischargeLoWarn" value="20.0" units="percent"/>
15:     <Qualifier name="DepthOfDischargeLoKeepout" value="10.0" units="percent"/>
16:     <Qualifier name="CycleLimit" value="500" units="cycles"/>
17:     <Variable  name="BatSOC"    kind="dischargeFraction"  format="FLOAT32"  units="percent"
description="percent of capacity discharged" />
18:     <Variable  name="BatTemp"   kind="temperature"       format="FLOAT32"  units="degC"
description="ambient temperature inside battery" />
19:     <Variable  name="BatCycleCount" kind="count"         format="INT16"    units="counts"
description="count of discharge/recharge cycles the device has experienced" />
20:     <Variable  name="BatCurrent" kind="electricalCurrent" format="FLOAT32"  units="A"
description="charging is positive; discharging is negative" />
21:     <Variable  name="BatUnregVoltage" kind="voltage"      format="FLOAT32"  units="V"
description="voltage at battery terminals" />
22:     <Variable  name="CurrentLimitIn" kind="electricalCurrent" format="FLOAT32"  units="A"
description="Limit on current flowing into battery" />
23:     <Variable  name="CurrentLimitOut" kind="electricalCurrent" format="FLOAT32"  units="A"
description="Limit on current flowing out of battery" />
24:     <Variable name="BatteryState" kind="mode" format="UINT08" >
25:       <Drange name="BatteryStateEnum" >
26:         <Option name="Offline" value="0"/>
27:         <Option name="Online" value="1"/>
28:         <Option name="Calibrate" value="2"/>
29:       </Drange>
30:     </Variable>
31:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
```

```

32: <Variable      kind="SubSeconds"      name="SubS"      units="Counts"      format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
33:
34: <Notification>
35:     <DataMsg name="BatSOH" description="state of health" id="1" msgArrival="PERIODIC">
36:         <Qualifier name="telemetryLevel" value="1"/>
37:         <VariableRef name="Time" />
38:         <VariableRef name="SubS" />
39:         <VariableRef name="BatteryState" />
40:         <VariableRef name="BatSOC" />
41:         <VariableRef name="BatTemp" />
42:         <VariableRef name="BatCycleCount" />
43:         <VariableRef name="BatCurrent" />
44:         <VariableRef name="BatUnregVoltage" />
45:     </DataMsg>
46: </Notification>
47:
48: <Command>
49:     <CommandMsg id="2" name="SetCurrentLimitIn">
50:         <VariableRef name="CurrentLimitIn"/>
51:     </CommandMsg>
52: </Command>
53:
54: <Command>
55:     <CommandMsg id="3" name="SetCurrentLimitOut">
56:         <VariableRef name="CurrentLimitOut"/>
57:     </CommandMsg>
58: </Command>
59:
60: <Command>
61:     <CommandMsg id="4" name="SetBatteryState">
62:         <VariableRef name="BatteryState"/>
63:     </CommandMsg>
64: </Command>
65: </Interface>
66:
67: <Interface name="CmpSafety" id="3">
68:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
69:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
70:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
71:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>

```

```

72:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
73:    <Variable    kind="SubSeconds"    name="SubS"    units="Counts"    format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
74:    <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
75:    <Request>
76:        <CommandMsg name="GetDeviceTemperature" id="1" />
77:        <DataReplyMsg name="DeviceTempReply" id="2">
78:            <VariableRef name="Time" />
79:            <VariableRef name="SubS" />
80:            <VariableRef name="DeviceTemperature"/>
81:        </DataReplyMsg>
82:    </Request>
83:    <Notification>
84:        <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
85:            <Qualifier name="telemetryLevel" value="1"/>
86:            <VariableRef name="Time" />
87:            <VariableRef name="SubS" />
88:            <VariableRef name="DeviceTemperature"/>
89:        </DataMsg>
90:    </Notification>
91: </Interface>
92:
93: </xTEDS>

```


File: sdm/app/test/DMTests/xTEDSRegTests/GPS_Full.h

```
1: #ifndef _FULLGPSRECEIVER_XTEDS_H
2: #define _FULLGPSRECEIVER_XTEDS_H
3:
4: #define _STRING_FULLGPSRECEIVER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
   \"FullGPSxTEDS \"\" \
8:   "version= \"2.0\">\" \
9:   "<Device name= \"FullGPSReceiver \" kind= \"sgr \" description= \"A GPS receiver that calculates
   position, velocity, and time internally \" />\" \
10:   \"\" \
11:   "<Interface name= \"GPSInterface \" id= \"1 \">\" \
12:   "<Qualifier name= \"headID \" value= \"0 \" />\" \
13:   "<Variable name= \"numVisibleSats \" kind= \"listSize \" format= \"INT16 \" description= \"Number
   of GPS satellites visible in this observation \" />\" \
14:   "<Variable name= \"recvTime \" kind= \"time \" format= \"INT32 \" units= \"s \" description=
   \"Receiver clock time since GPS epoch \">\" \
15:   "<Qualifier name= \"timeFrame \" value= \"GPST1 \" />\" \
16:   "</Variable>\" \
17:   "<Variable name= \"PRNs \" kind= \"ID \" format= \"INT16 \" length= \"28 \" description= \"list of
   PRN numbers of the visible GPS satellites \">\" \
18:   "<Qualifier name= \"representation \" value= \"array \" />\" \
19:   "</Variable>\" \
20:   "<Variable name= \"PRN \" kind= \"ID \" format= \"INT16 \" description= \"PRN of a single GPS sat
   generating a NAV message \" />\" \
21:   "<Variable name= \"Prange \" kind= \"distance \" format= \"FLOAT32 \" length= \"28 \" units= \"km
   \" description= \"Pseudorange measurements for each visible GPS satellite \">\" \
22:   "<Qualifier name= \"representation \" value= \"array \" />\" \
23:   "</Variable>\" \
24:   "<Variable name= \"epochTime \" kind= \"epoch \" format= \"INT16 \" length= \"6 \" description=
   \"YMDHMS definition of the current GPS epoch time \">\" \
25:   "<Qualifier name= \"representation \" value= \"array \" />\" \
26:   "</Variable>\" \
27:   "<Variable name= \"nav \" kind= \"navMessage \" format= \"FLOAT32 \" length= \"28 \"
   description= \"Nav message from this satellite: ephemeris, clock corrections \">\" \
28:   "<Qualifier name= \"representation \" value= \"array \" />\" \
29:   "</Variable>\" \
30:   "<Variable name= \"gpst \" kind= \"time \" format= \"INT32 \" units= \"s \" description= \"Current
   calculated GPS time (UTC) \">\" \
```

```

31: "<Qualifier name= \"timeFrame \" value= \"UTC \" />\" \
32: "</Variable>\" \
33: "<Variable name= \"R \" kind= \"distance \" format= \"FLOAT32 \" units= \"km \" length= \"3 \"
description= \"Current position of the receiver in ECEF \">\" \
34: "<Qualifier name= \"representation \" value= \"vector \" />\" \
35: "<Qualifier name= \"frameMeasured \" value= \"ITRF \" />\" \
36: "<Qualifier name= \"frameResolved \" value= \"ITRF \" />\" \
37: "</Variable>\" \
38: "<Variable name= \"V \" kind= \"velocity \" format= \"FLOAT32 \" units= \"km_s \" length= \"3 \"
description= \"Current velocity vector of the receiver in ECEF \">\" \
39: "<Qualifier name= \"representation \" value= \"vector \" />\" \
40: "<Qualifier name= \"frameMeasured \" value= \"ITRF \" />\" \
41: "<Qualifier name= \"frameResolved \" value= \"ITRF \" />\" \
42: "</Variable>\" \
43: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
44: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
45: "<Notification>\" \
46: "<DataMsg name= \"GPSSatPrange \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"1 \">\" \
47: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
48: "<VariableRef name= \"numVisibleSats \" />\" \
49: "<VariableRef name= \"recvTime \" />\" \
50: "<VariableRef name= \"PRNs \" />\" \
51: "<VariableRef name= \"Prange \" />\" \
52: "</DataMsg>\" \
53: "</Notification>\" \
54: "<Notification>\" \
55: "<DataMsg name= \"GPSNavMessage \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"2 \">\" \
56: "<VariableRef name= \"PRN \" />\" \
57: "<VariableRef name= \"epochTime \" />\" \
58: "<VariableRef name= \"nav \" />\" \
59: "</DataMsg>\" \
60: "</Notification>\" \
61: "<Notification>\" \
62: "<DataMsg name= \"SCPosition \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"3 \">\" \
63: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
64: "<VariableRef name= \"Time \" />\" \
65: "<VariableRef name= \"SubS \" />\" \
66: "<VariableRef name= \"gpst \" />\" \
67: "<VariableRef name= \"R \" />\" \
68: "</DataMsg>\" \
69: "</Notification>\" \

```

```

70: "<Notification>" \
71: "<DataMsg name= \"SCVelocity \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"4 \">>" \
72: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
73: "<VariableRef name= \"Time \"/>" \
74: "<VariableRef name= \"SubS \"/>" \
75: "<VariableRef name= \"gpst \" />" \
76: "<VariableRef name= \"V \" />" \
77: "</DataMsg>" \
78: "</Notification>" \
79: "<Notification>" \
80: "<DataMsg name= \"GPSTime \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"5 \">>" \
81: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
82: "<VariableRef name= \"gpst \" />" \
83: "</DataMsg>" \
84: "</Notification>" \
85: "</Interface>" \
86: "" \
87: "<Interface name= \"DevPwr \" id= \"2 \">>" \
88: "<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \"/>" \
89: "<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \"/>" \
90: "<Qualifier name= \"CurrentHiWarning \" value= \"3.5 \" units= \"A \"/>" \
91: "<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \"/>" \
92: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
93: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
94: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \">>" \
95: "<Drange name= \"DevPwrStateEnum \">>" \
96: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />" \
97: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
98: "</Drange>" \
99: "</Variable>" \
100: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \">>" \
101: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to DevPwrStateEnumeration. \">>" \
102: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />" \
103: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
104: "</Drange>" \
105: "</Variable>" \
106: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length= \"2 \"/>" \

```

```

107: "<Command>" \
108: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \">" \
109: "<VariableRef name= \"DevPwrStateSet \" />" \
110: "</CommandMsg>" \
111: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \">" \
112: "<VariableRef name= \"Time \" />" \
113: "<VariableRef name= \"SubS \" />" \
114: "<VariableRef name= \"DevPwrState \" />" \
115: "<VariableRef name= \"DevPwrStateSet \" />" \
116: "</FaultMsg>" \
117: "</Command>" \
118: "<Notification>" \
119: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">" \
120: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
121: "<VariableRef name= \"Time \" />" \
122: "<VariableRef name= \"SubS \" />" \
123: "<VariableRef name= \"DevPwrState \" />" \
124: "<VariableRef name= \"DevPwrStateSet \" />" \
125: "</DataMsg>" \
126: "</Notification>" \
127: "<Request>" \
128: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
129: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">" \
130: "<VariableRef name= \"modePowers \" />" \
131: "</DataReplyMsg>" \
132: "</Request>" \
133: "</Interface>" \
134: "" \
135: "<Interface name= \"CmpSOH \" id= \"3 \">" \
136: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \" />" \
137: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \" />" \
138: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \" />" \
139: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \" />" \
140: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
141: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
142: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \" />" \
143: "<Request>" \
144: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />" \
145: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">" \

```

```

146: "<VariableRef name= \"Time \" />" \
147: "<VariableRef name= \"SubS \" />" \
148: "<VariableRef name= \"DeviceTemperature \" />" \
149: "</DataReplyMsg>" \
150: "</Request>" \
151: "<Notification>" \
152: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
153: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
154: "<VariableRef name= \"Time \" />" \
155: "<VariableRef name= \"SubS \" />" \
156: "<VariableRef name= \"DeviceTemperature \" />" \
157: "</DataMsg>" \
158: "</Notification>" \
159: "</Interface>" \
160: "</xTEDS>" \
161: ""
162:
163: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/MagTorqueRod.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS           xTEDS02.xsd"
name="MagneticTorqueRodxTeds"
4:   version="2.0">
5:   <Device name="BasicMagneticTorqueRod" kind="mtr" description="Basic (on/off + polarity)
magnetic torquer" />
6:   <Interface name="MagTorquerBasic" id="1">
7:     <Qualifier name="headID" value="0"/>
8:     <Variable name="polarity" kind="polarity" units="0_1" format="UINT08">
9:       <Drange name="PolarityEnum">
10:        <Option name="North" value="0" description="North polarity of device dipole" />
11:        <Option name="South" value="1" description="South polarity of device dipole" />
12:      </Drange>
13:    </Variable>
14:    <Command>
15:      <CommandMsg name="MTROnCmd" id="1">
16:        <VariableRef name="polarity" />
17:      </CommandMsg>
18:    </Command>
19:    <Command>
20:      <CommandMsg name="MTROffCmd" id="2" />
21:    </Command>
22:  </Interface>
23:
24:  <Interface name="DevPwr" id="2">
25:    <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
26:    <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
27:    <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
28:    <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
29:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
30:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
31:    <Variable name="DevPwrState" kind="Power_State" format="UINT08">
32:      <Drange name="DevPwrStateEnum">
33:        <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
34:        <Option name="DevPwrON" value="1" description="Device may draw full power." />
35:      </Drange>
36:    </Variable>
```

```

37: <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
38:     <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
39:     <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
40:     <Option name="DevPwrON" value="1" description="Device may draw full power." />
41: </Drange>
42: </Variable>
43: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
44: <Command>
45:     <CommandMsg name="DevPwrSetState" id="1">
46:         <VariableRef name="DevPwrStateSet" />
47:     </CommandMsg>
48:     <FaultMsg name="DevPwrStateNotSet" id="2">
49:         <VariableRef name="Time" />
50:         <VariableRef name="SubS" />
51:         <VariableRef name="DevPwrState" />
52:         <VariableRef name="DevPwrStateSet" />
53:     </FaultMsg>
54: </Command>
55: <Notification>
56:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
57:         <Qualifier name="telemetryLevel" value="1"/>
58:         <VariableRef name="Time" />
59:         <VariableRef name="SubS" />
60:         <VariableRef name="DevPwrState" />
61:         <VariableRef name="DevPwrStateSet" />
62:     </DataMsg>
63: </Notification>
64: <Request>
65:     <CommandMsg name="getPowerInMode" id="4" />
66:     <DataReplyMsg name="powerInMode" id="5">
67:         <VariableRef name="modePowers"/>
68:     </DataReplyMsg>
69: </Request>
70: </Interface>
71:
72: <Interface name="CmpSOH" id="3">
73:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
74:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
75:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
76:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>

```

```

77: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
78:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
79: <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
80: <Request>
81:     <CommandMsg name="GetDeviceTemperature" id="1" />
82:     <DataReplyMsg name="DeviceTempReply" id="2">
83:         <VariableRef name="Time" />
84:         <VariableRef name="SubS" />
85:         <VariableRef name="DeviceTemperature"/>
86:     </DataReplyMsg>
87: </Request>
88: <Notification>
89:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
90:         <Qualifier name="telemetryLevel" value="1"/>
91:         <VariableRef name="Time" />
92:         <VariableRef name="SubS" />
93:         <VariableRef name="DeviceTemperature"/>
94:     </DataMsg>
95: </Notification>
96: </Interface>
97: </xTEDS>

```


File: sdm/app/test/DMTests/xTEDSRegTests/ThreeAxisMagnetometer.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS                                xTEDS02.xsd"
name="ThreeAxisMagnetometerXTEDS"
4:   version="2.0">
5:   <Device name="Three_axis_magnetometer" kind="mag" description="3-axis output digital
magnetometer" />
6:   <Interface name="ThreeAxisMagBasic" id="1">
7:     <Qualifier name="headID" value="0"/>
8:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
9:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
10:    <Variable name="BVector" kind="magFlux" units="weber" format="FLOAT32" length="3"
description="Magnetic flux observed at this device" >
11:      <Qualifier name="representation" value="vector" />
12:      <Qualifier name="FrameMeasured" value="DVF" />
13:      <Qualifier name="FrameResolved" value="DVF" />
14:    </Variable>
15:    <Notification>
16:      <DataMsg name="MagField" msgArrival="PERIODIC" msgRate="1" id="1">
17:        <Qualifier name="telemetryLevel" value="1"/>
18:        <VariableRef name="Time"/>
19:        <VariableRef name="SubS"/>
20:        <VariableRef name="BVector" />
21:      </DataMsg>
22:    </Notification>
23:  </Interface>
24:
25:  <Interface name="DevPwr" id="2">
26:    <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
27:    <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
28:    <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
29:    <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
30:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
31:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
32:    <Variable name="DevPwrState" kind="Power_State" format="UINT08">
33:      <Drange name="DevPwrStateEnum">
34:        <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
```

```

35:     <Option name="DevPwrON" value="1" description="Device may draw full power." />
36:   </Drange>
37: </Variable>
38: <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
39:     <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
40:       <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
41:       <Option name="DevPwrON" value="1" description="Device may draw full power." />
42:     </Drange>
43: </Variable>
44: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
45: <Command>
46:   <CommandMsg name="DevPwrSetState" id="1">
47:     <VariableRef name="DevPwrStateSet" />
48:   </CommandMsg>
49:   <FaultMsg name="DevPwrStateNotSet" id="2">
50:     <VariableRef name="Time" />
51:     <VariableRef name="SubS" />
52:     <VariableRef name="DevPwrState" />
53:     <VariableRef name="DevPwrStateSet" />
54:   </FaultMsg>
55: </Command>
56: <Notification>
57:   <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
58:     <Qualifier name="telemetryLevel" value="1"/>
59:     <VariableRef name="Time" />
60:     <VariableRef name="SubS" />
61:     <VariableRef name="DevPwrState" />
62:     <VariableRef name="DevPwrStateSet" />
63:   </DataMsg>
64: </Notification>
65: <Request>
66:   <CommandMsg name="getPowerInMode" id="4" />
67:   <DataReplyMsg name="powerInMode" id="5">
68:     <VariableRef name="modePowers"/>
69:   </DataReplyMsg>
70: </Request>
71: </Interface>
72:
73: <Interface name="CmpSOH" id="3">
74:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>

```

```

75: <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
76: <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
77: <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
78: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
79:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
80: <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
81: <Request>
82:     <CommandMsg name="GetDeviceTemperature" id="1" />
83:     <DataReplyMsg name="DeviceTempReply" id="2">
84:         <VariableRef name="Time" />
85:         <VariableRef name="SubS" />
86:         <VariableRef name="DeviceTemperature"/>
87:     </DataReplyMsg>
88: </Request>
89: <Notification>
90:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
91:         <Qualifier name="telemetryLevel" value="1"/>
92:         <VariableRef name="Time" />
93:         <VariableRef name="SubS" />
94:         <VariableRef name="DeviceTemperature"/>
95:     </DataMsg>
96: </Notification>
97: </Interface>
98: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/TelemetryHandler.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd"
name="TelemetryHandlerXTEDS" description="TelemetryHandler xTEDS" version="1.0">
4:
5:         <Application      name="TelemetryHandler"      kind="CommunicationsFlightSoftware"
description="Communications Subsystem Telemetry Handler"/>
6:
7:     <Interface name="TelemetryHandlerInterface" id="1" description="Basic Telemetry Manager
interface">
8:
9:         <Variable name="ComponentSensorID"      kind="tbd"              format="UINT32"/>
10:        <Variable name="MsgID"                  kind="tbd" rangeMin="1" rangeMax="255"
format="UINT08"/>
11:        <Variable name="InterfaceID"            kind="tbd" rangeMin="0" rangeMax="255"
format="UINT08"/>
12:        <Variable name="ComponentTelemetryLevel" kind="tbd" rangeMin="0" rangeMax="4"
format="UINT08"/>
13:        <Variable name="MsgTelemetryLevel"      kind="tbd" rangeMin="1" rangeMax="3"
format="UINT08"/>
14:        <Variable name="SendX"                  kind="tbd"              format="UINT08"/>
15:        <Variable name="OfY"                    kind="tbd"              format="UINT08"/>
16:
17:    <Command>
18:        <CommandMsg name="RestoreMsgTlmLevel" id="002" description="Restore the telemetry level
of the message to default level">
19:            <VariableRef name="ComponentSensorID"/>
20:            <VariableRef name="MsgID"/>
21:            <VariableRef name="InterfaceID"/>
22:        </CommandMsg>
23:    </Command>
24:
25:    <Command>
26:        <CommandMsg name="SetComponentTlmLevel" id="003" description="Set the telemetry level
of this component">
27:            <VariableRef name="ComponentSensorID"/>
28:            <VariableRef name="ComponentTelemetryLevel"/>
29:        </CommandMsg>
30:    </Command>
31:
```

```

32: <Command>
33:     <CommandMsg name="SetComponentTlmLevelToDefault" id="004" description="Set the
telemetry level of this component to default">
34:         <VariableRef name="ComponentSensorID"/>
35:     </CommandMsg>
36: </Command>
37:
38: <Command>
39:     <CommandMsg name="SetDefaultTlmLevel" id="005" description="Set the overall telemetry
level of the SYSTEM">
40:         <VariableRef name="ComponentTelemetryLevel"/>
41:     </CommandMsg>
42: </Command>
43:
44: <Command>
45:     <CommandMsg name="SetMsgTlmLevel" id="006" description="Set the telemetry level of the
message">
46:         <VariableRef name="ComponentSensorID"/>
47:         <VariableRef name="MsgID"/>
48:         <VariableRef name="InterfaceID"/>
49:         <VariableRef name="MsgTelemetryLevel"/>
50:     </CommandMsg>
51: </Command>
52:
53: <Command>
54:     <CommandMsg name="SetMsgDownlinkRate" id="007" description="Set the rate at which this
message will be downlinked">
55:         <VariableRef name="ComponentSensorID"/>
56:         <VariableRef name="MsgID"/>
57:         <VariableRef name="InterfaceID"/>
58:         <VariableRef name="SendX"/>
59:         <VariableRef name="OfY"/>
60:     </CommandMsg>
61: </Command>
62:
63: </Interface>
64:
65: <Interface name="TelemetryHandlerStatusInterface" id="2">
66:
67:     <Variable name="TelemetryHandlerStatus" kind="Status" format="UINT08">
68:         <Drange name="TelemetryHandlerStatusEnum">

```

```

69:      <Option value="0" name="NOT_INITIALIZED"/><!-- The TelemetryHandler has not been
successfully initialized -->
70:      <Option value="1" name="INITIALIZING"/>
71:      <Option value="2" name="RUNNING"/><!-- The TelemetryHandler is initialized and is running
-->
72:      <Option value="3" name="TERMINATING"/><!-- The TelemetryHandler is shutting down -->
73:  </Drange>
74:  </Variable>
75:
76:      <Variable name="SystemTelemetryLevel" kind="tbd" rangeMin="0" rangeMax="4"
format="UINT08"/>
77:  <Variable name="DataMsgsReceived" kind="tbd" format="UINT32"/>
78:  <Variable name="DataMsgsForwarded" kind="tbd" format="UINT32"/>
79:  <Variable name="ComponentsRegistered" kind="tbd" format="UINT16"/>
80:  <Variable name="DataMsgsRegistered" kind="tbd" format="UINT16"/>
81:  <Variable name="DataMsgsSubscribed" kind="tbd" format="UINT16"/>
82:  <Variable name="CommandsReceived" kind="tbd" format="UINT32"/>
83:  <Variable name="CommandsAccepted" kind="tbd" format="UINT32"/>
84:  <Variable name="CommandsRejected" kind="tbd" format="UINT32"/>
85:
86:  <Notification>
87:    <DataMsg name="TelemetryHandlerStatusMsg" id="001" msgArrival="EVENT">
88:      <Qualifier value="1" name="telemetryLevel"/>
89:      <VariableRef name="TelemetryHandlerStatus"/>
90:      <VariableRef name="ComponentsRegistered"/>
91:      <VariableRef name="DataMsgsReceived"/>
92:      <VariableRef name="DataMsgsForwarded"/>
93:      <VariableRef name="DataMsgsRegistered"/>
94:      <VariableRef name="DataMsgsSubscribed"/>
95:      <VariableRef name="CommandsReceived"/>
96:      <VariableRef name="CommandsAccepted"/>
97:      <VariableRef name="CommandsRejected"/>
98:      <VariableRef name="SystemTelemetryLevel"/>
99:    </DataMsg>
100:  </Notification>
101:
102: </Interface>
103:
104: <Interface name="ICSDebugInterface" id="3">
105:
106:  <!--
107:    Note: DebugLevel is a bit field with the following assigned values:

```

```

108:     DEBUG_ENTRY_AND_EXIT = 0x01,
109:     DEBUG_ENTRY_PARAMETERS = 0x02,
110:     DEBUG_EXIT_PARAMETERS = 0x04,
111:     DEBUG_LEVEL_LOW      = 0x08,
112:     DEBUG_LEVEL_MEDIUM   = 0x10,
113:     DEBUG_LEVEL_HIGH     = 0x20.
114:     The values are OR'd to determine the debug information to be logged.
115: -->
116:
117:     <Variable name="DebugLevel" kind="tbd" format="UINT16"/>
118:     <Variable name="CurrentDebugLevel" kind="tbd" format="UINT16"/>
119:
120:     <Variable name="SetDebugLevelReceived" kind="tbd" format="UINT16"/>
121:     <Variable name="SetDebugLevelAccepted" kind="tbd" format="UINT16"/>
122:     <Variable name="SetDebugLevelSuccess" kind="tbd" format="UINT16"/>
123:     <Variable name="SetDebugLevelFailure" kind="tbd" format="UINT16"/>
124:
125:     <Command>
126:         <CommandMsg name="SetDebugLevel" id="001" description="Set the debug log verbosity
level">
127:             <VariableRef name="DebugLevel"/>
128:         </CommandMsg>
129:     </Command>
130:
131:     <Command>
132:         <CommandMsg name="SendDebugStatus" id="002"/>
133:     </Command>
134:
135:     <Notification>
136:         <DataMsg name="DebugStatus" id="03" msgArrival="EVENT">
137:             <Qualifier value="1" name="telemetryLevel"/>
138:             <VariableRef name="CurrentDebugLevel"/>
139:             <VariableRef name="SetDebugLevelReceived"/>
140:             <VariableRef name="SetDebugLevelAccepted"/>
141:             <VariableRef name="SetDebugLevelSuccess"/>
142:             <VariableRef name="SetDebugLevelFailure"/>
143:         </DataMsg>
144:     </Notification>
145:
146: </Interface>
147:

```

```
148: <Interface name="TaskControlInterface" id="4">
149:
150:   <Command>
151:     <CommandMsg name="DestroyTask" id="001"/>
152:   </Command>
153:
154:   <!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->
155:   <!-- Other possible requests include GetPriority, and GetHeartBeatCount -->
156:
157: </Interface>
158:
159:
160: </xTEDS>
161:
```


File: sdm/app/test/DMTTests/xTEDSRegTests/Makefile

```
1: include ../../../../Makefile.defs
2:
3:
4: .PHONY:    all clean distclean
5:
6: all: xTEDSRegTest
7:
8: xTEDSRegTest: xTEDSRegTest.o
9:  $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -L../../common -lSDM
10:
11: %.o: %.cpp
12:  $(CXX) $(CXXFLAGS) -c $<
13:
14: clean:
15: rm -f *.o
16:
17: distclean: clean
18: rm -f xTEDSRegTest *~
```

File: sdm/app/test/DMTests/xTEDSRegTests/BIT.h

```
1: #ifndef _EXAMPLE_XTEDS_H
2: #define _EXAMPLE_XTEDS_H
3:
4: #define _STRING_EXAMPLE_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
7:   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
8:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
9:   name= \"BITxTEDS\" description= \"Built-In Test interface templates\" version= \"0.1\">\" \
10:   \"\" \
11:   \"<Application name= \"Example\" kind= \"REFERENCE_ONLY\" description= \"A null application.
12:   The purpose of this xTEDS is to define templates for BIT interfaces.\">\" \
13:   \"\" \
14:   \"<Interface name= \"FunctionalTest\" id= \"1\" description= \"The Functional Built-In Test is a basic
15:   functionality test. Typically initiated manually\">\" \
16:   \"\" \
17:   \"<Variable name= \"functionalTestResultCode\" kind= \"STATUS\" format= \"UINT08\"
18:   description= \"The result of running a Built-In Functional Test\">\" \
19:   \"<Drange name= \"functionalTestResultCodes\" description= \"An enumeration of all test results\">\"
20:   \
21:   \"<Option value= \"0\" name= \"NOT_EXECUTED\" description= \"The test has not been executed.
22:   \">\" \
23:   \"<Option value= \"1\" name= \"SUCCESS\" description= \"The test ran successfully.\">\" \
24:   \"<Option value= \"2\" name= \"FAILURE\" description= \"The test failed.\">\" \
25:   \"<!-- Note: FAILURE is a placeholder for defining specific failures. Typically all diagnosable failures
26:   are enumerated. -->\" \
27:   \"</Drange>\" \
28:   \"</Variable>\" \
29:   \"\" \
30:   \"<Variable kind= \"Time\" name= \"functionalTestSeconds\" format= \"UINT32\" units= \"Seconds
31:   \" description= \"Time test was last executed.\">\" \
32:   \"<Variable kind= \"SubSeconds\" name= \"functionalTestSubSeconds\" units= \"Counts\" format=
33:   \"UINT32\" scaleFactor= \".0001\" scaleUnits= \"Seconds\"\" \
34:   \"description= \"Time test was last executed.\">\" \
35:   \"\" \
36:   \"<Request>\" \
37:   \"<CommandMsg name= \"runFunctionalTest\" id= \"001\" description= \"Run the Functional BIT
38:   and report results.\">\" \
39:   \"<DataReplyMsg name= \"functionalTestResult\" id= \"002\">\" \
40:   \"<VariableRef name= \"functionalTestResultCode\">\" \
41:   \"</DataReplyMsg>\" \
```

```

31: "</Request>" \
32: "" \
33: "<Request>" \
34: "<CommandMsg name= \"getLastFunctionalTestResult \" id= \"003 \" description= \"Run the
Functional BIT and report results. \"/>" \
35: "<DataReplyMsg name= \"lastFunctionalTestResult \" id= \"004 \"/>" \
36: "<VariableRef name= \"functionalTestSeconds \"/>" \
37: "<VariableRef name= \"functionalTestSubSeconds \"/>" \
38: "<VariableRef name= \"functionalTestResultCode \"/>" \
39: "</DataReplyMsg>" \
40: "</Request>" \
41: "" \
42: "<Notification>" \
43: "<DataMsg name= \"functionalTestResultChange \" id= \"005 \" msgArrival= \"EVENT \"
description= \"Functional Test result has changed. \"/>" \
44: "<VariableRef name= \"functionalTestSeconds \"/>" \
45: "<VariableRef name= \"functionalTestSubSeconds \"/>" \
46: "<VariableRef name= \"functionalTestResultCode \"/>" \
47: "</DataMsg>" \
48: "</Notification>" \
49: "" \
50: "</Interface>" \
51: "" \
52: "<Interface id= \"2 \" name= \"PerformanceTest \" description= \"The Performance Built-In Test is a
more extensive operational test. Typically initiated manually. \"/>" \
53: "" \
54: "<Variable name= \"performanceTestResultCode \" kind= \"STATUS \" format= \"UINT08 \"
description= \"The result of running a Built-In Performance Test \"/>" \
55: "<Drange name= \"performanceTestResultCodes \" description= \"An enumeration of all test results
\"/>" \
56: "<Option value= \"0 \" name= \"NOT_EXECUTED \" description= \"The test has not been executed.
\"/>" \
57: "<Option value= \"1 \" name= \"SUCCESS \" description= \"The test ran successfully. \"/>" \
58: "<Option value= \"2 \" name= \"FAILURE \" description= \"The test failed. \"/>" \
59: "<!-- Note: FAILURE is a placeholder for defining specific failures. Typically all diagnosable failures
are enumerated. -->" \
60: "</Drange>" \
61: "</Variable>" \
62: "" \
63: "<Variable kind= \"Time \" name= \"performanceTestSeconds \" format= \"UINT32 \" units=
\"Seconds \" description= \"Time test was last executed. \"/>" \
64: "<Variable kind= \"SubSeconds \" name= \"performanceTestSubSeconds \" units= \"Counts \"
format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" \

```

```

65: "description= \"Time test was last executed. \"/>" \
66: "" \
67: "<Request>" \
68: "<CommandMsg name= \"runPerformanceTest \" id= \"001 \" description= \"Run the Performance
BIT and report results. \"/>" \
69: "<DataReplyMsg name= \"performanceTestResult \" id= \"002 \"/>" \
70: "<VariableRef name= \"performanceTestResultCode \"/>" \
71: "</DataReplyMsg>" \
72: "</Request>" \
73: "" \
74: "<Request>" \
75: "<CommandMsg name= \"getLastPerformanceTestResult \" id= \"003 \" description= \"Run the
Performance BIT and report results. \"/>" \
76: "<DataReplyMsg name= \"lastPerformanceTestResult \" id= \"004 \"/>" \
77: "<VariableRef name= \"performanceTestSeconds \"/>" \
78: "<VariableRef name= \"performanceTestSubSeconds \"/>" \
79: "<VariableRef name= \"performanceTestResultCode \"/>" \
80: "</DataReplyMsg>" \
81: "</Request>" \
82: "" \
83: "<Notification>" \
84: "<DataMsg name= \"performanceTestResultChange \" id= \"005 \" msgArrival= \"EVENT \"
description= \"Performance Test result has changed. \"/>" \
85: "<VariableRef name= \"performanceTestSeconds \"/>" \
86: "<VariableRef name= \"performanceTestSubSeconds \"/>" \
87: "<VariableRef name= \"performanceTestResultCode \"/>" \
88: "</DataMsg>" \
89: "</Notification>" \
90: "" \
91: "</Interface>" \
92: "" \
93: "<Interface id= \"3 \" name= \"OnOrbitCheckout \" description= \"On-Orbit Checkout verifies
operational readiness. Autonomously initiated by OOCE. \"/>" \
94: "" \
95: "<Variable name= \"onOrbitCheckoutResultCode \" kind= \"STATUS \" format= \"UINT08 \"
description= \"The result of running an On-Orbit Checkout \"/>" \
96: "<Drange name= \"onOrbitCheckoutResultCodes \" description= \"An enumeration of all test results
\"/>" \
97: "<Option value= \"0 \" name= \"NOT_EXECUTED \" description= \"The On-Orbit Checkout has not
been executed. \"/>" \
98: "<Option value= \"1 \" name= \"SUCCESS \" description= \"The On-Orbit Checkout ran successfully.
\"/>" \
99: "<Option value= \"2 \" name= \"FAILURE \" description= \"The On-Orbit Checkout failed. \"/>" \

```

```

100: "<!-- Note: FAILURE is a placeholder for defining specific failures. Typically all diagnosable  
failures are enumerated. -->" \
101: "</Drange>" \
102: "</Variable>" \
103: "" \
104: "<Variable kind= \"Time \" name= \"onOrbitCheckoutSeconds \" format= \"UINT32 \" units=  
\"Seconds \" description= \"TimeOn-Orbit Checkout was last executed. \"/>" \
105: "<Variable kind= \"SubSeconds \" name= \"onOrbitCheckoutSubSeconds \" units= \"Counts \"  
format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \"\" \
106: "description= \"Time On-Orbit Checkout was last executed. \"/>" \
107: "" \
108: "<Request>" \
109: "<CommandMsg name= \"runOnOrbitCheckout \" id= \"001 \" description= \"Run the On-Orbit  
Checkout and report results. \"/>" \
110: "<DataReplyMsg name= \"onOrbitCheckoutResult \" id= \"002 \"/>" \
111: "<VariableRef name= \"onOrbitCheckoutResultCode \"/>" \
112: "</DataReplyMsg>" \
113: "</Request>" \
114: "" \
115: "<Request>" \
116: "<CommandMsg name= \"getLastOnOrbitCheckoutResult \" id= \"003 \" description= \"Run the  
On-Orbit Checkout and report results. \"/>" \
117: "<DataReplyMsg name= \"lastOnOrbitCheckoutResult \" id= \"004 \"/>" \
118: "<VariableRef name= \"onOrbitCheckoutSeconds \"/>" \
119: "<VariableRef name= \"onOrbitCheckoutSubSeconds \"/>" \
120: "<VariableRef name= \"onOrbitCheckoutResultCode \"/>" \
121: "</DataReplyMsg>" \
122: "</Request>" \
123: "" \
124: "<Notification>" \
125: "<DataMsg name= \"onOrbitCheckoutResultChange \" id= \"005 \" msgArrival= \"EVENT \"  
description= \"On-Orbit Checkout result has changed. \"/>" \
126: "<VariableRef name= \"onOrbitCheckoutSeconds \"/>" \
127: "<VariableRef name= \"onOrbitCheckoutSubSeconds \"/>" \
128: "<VariableRef name= \"onOrbitCheckoutResultCode \"/>" \
129: "</DataMsg>" \
130: "</Notification>" \
131: "" \
132: "</Interface>" \
133: "" \
134: "</xTEDS>" \
135: ""

```

136:
137: #endif

File: sdm/app/test/DMTests/xTEDSRegTests/RWheelSingle.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="Reaction_Wheel_XTEDS"
4:   version="2.1">
5:   <Device name="Single_Reaction_Wheel" kind="rWheel" description="An xTeds for a single
reaction wheel using the 2.0 xTeds schema and the CDD"/>
6:
7:   <Interface name="RWSingleInterface" id="1">
8:     <Qualifier name="MaxMomentum" value="5.0" units="Nms" />
9:     <Qualifier name="NominalMomentum" value="2.0" units="Nms" />
10:    <Qualifier name="TorqueLossMomentum" value="1.0" units="Nms"/>
11:    <Qualifier name="MaxTorqueAtMaxSpeed" value="10.0" units="Nm" />
12:    <Qualifier name="MaxTorqueAtNominalSpeed" value="5.0" units="Nm"/>
13:    <Qualifier name="TimeConstant" value="0.1" units="s" />
14:    <Qualifier name="IdlePower" value="1.0" units="W"/>
15:    <Qualifier name="MaxPower" value="10.0" units="W"/>
16:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
17:    <Variable   kind="SubSeconds"   name="SubS"   units="Counts"   format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
18:    <Variable name="opMode" kind="mode" format="UINT16" >
19:      <Drange name="enumModes" >
20:        <Option name="idle" value="0"/>
21:        <Option name="operating" value="1"/>
22:      </Drange>
23:    </Variable>
24:    <Variable name="dataQuality" kind="DataQuality" format="UINT08">
25:      <Drange name="DataQualityEnum">
26:        <Option name="dataBad" value="0" description="data is garbage or NaN." />
27:        <Option name="dataPoor" value="1" description="data quality is poor." />
28:        <Option name="dataGood" value="2" description="data is good." />
29:      </Drange>
30:    </Variable>
31:    <Variable name="commandedTorque" kind="torque" format="FLOAT32" units="Nm"
description="Magnitude of torque requested of this device" />
32:    <Variable name="currentMomentum" kind="angularMomentum" format="FLOAT32"
units="Nms" description="The current momentum of this wheel." />
33:    <Variable name="satPercentage" kind="saturationLevel" format="FLOAT32" units="percent"
description="The current percentage of saturation of this wheel." />
```

```

34: <Command>
35:     <!-- note: the name of a command uniquely identifies it within the dictionary -->
36:     <CommandMsg name="SingleAxisTorqueCmd" id="1">
37:         <VariableRef name="commandedTorque" />
38:     </CommandMsg>
39: </Command>
40: <!-- note: the "kind" attribute is the key identifier of a data element in the dictionary -->
41: <Notification>
42:     <DataMsg name="wheelSatLevel" msgArrival="PERIODIC" msgRate="1" id="2">
43:         <Qualifier name="telemetryLevel" value="1"/>
44:         <VariableRef name="Time" />
45:         <VariableRef name="SubS" />
46:         <VariableRef name="dataQuality" />
47:         <VariableRef name="satPercentage" />
48:     </DataMsg>
49: </Notification>
50: <Notification>
51:     <DataMsg name="MomentumCurrent" msgArrival="PERIODIC" msgRate="10" id="3">
52:         <Qualifier name="telemetryLevel" value="1"/>
53:         <VariableRef name="Time" />
54:         <VariableRef name="SubS" />
55:         <VariableRef name="dataQuality" />
56:         <VariableRef name="currentMomentum" />
57:     </DataMsg>
58: </Notification>
59: <Command>
60:     <CommandMsg name="setOpMode" id="4">
61:         <VariableRef name="opMode"/>
62:     </CommandMsg>
63: </Command>
64: <Notification>
65:     <DataMsg name="opModeChanged" id="5" msgArrival="EVENT">
66:         <Qualifier name="telemetryLevel" value="1"/>
67:         <VariableRef name="opMode"/>
68:     </DataMsg>
69: </Notification>
70: </Interface>
71:
72: <Interface name="DevPwr" id="4">
73:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
74:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>

```



```

75:    <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
76:    <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
77:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
78:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
79:    <Variable name="DevPwrState" kind="Power_State" format="UINT08">
80:        <Drange name="DevPwrStateEnum">
81:            <Option name="DevPwrOFF" value="0" description="All power to device is turned off."
/>
82:            <Option name="DevPwrON" value="1" description="Device may draw full power." />
83:        </Drange>
84:    </Variable>
85:    <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
86:        <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
87:            <Option name="DevPwrOFF" value="0" description="All power to device is turned off."
/>
88:            <Option name="DevPwrON" value="1" description="Device may draw full power." />
89:        </Drange>
90:    </Variable>
91:    <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
92:    <Command>
93:        <CommandMsg name="DevPwrSetState" id="1">
94:            <VariableRef name="DevPwrStateSet" />
95:        </CommandMsg>
96:        <FaultMsg name="DevPwrStateNotSet" id="2">
97:            <VariableRef name="Time" />
98:            <VariableRef name="SubS" />
99:            <VariableRef name="DevPwrState" />
100:            <VariableRef name="DevPwrStateSet" />
101:        </FaultMsg>
102:    </Command>
103:    <Notification>
104:        <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
105:            <Qualifier name="telemetryLevel" value="1"/>
106:            <VariableRef name="Time" />
107:            <VariableRef name="SubS" />
108:            <VariableRef name="DevPwrState" />
109:            <VariableRef name="DevPwrStateSet" />
110:        </DataMsg>
111:    </Notification>
112:    <Request>

```

```

113:         <CommandMsg name="getPowerInMode" id="4" />
114:         <DataReplyMsg name="powerInMode" id="5">
115:             <VariableRef name="modePowers"/>
116:         </DataReplyMsg>
117:     </Request>
118:
119: </Interface>
120:
121: <Interface name="CmpSOH" id="5">
122:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
123:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
124:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
125:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
126:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
127:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
128:     <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32"
units="degC" />
129:     <Request>
130:         <CommandMsg name="GetDeviceTemperature" id="1" />
131:         <DataReplyMsg name="DeviceTempReply" id="2">
132:             <VariableRef name="Time" />
133:             <VariableRef name="SubS" />
134:             <VariableRef name="DeviceTemperature"/>
135:         </DataReplyMsg>
136:     </Request>
137:     <Notification>
138:         <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
139:             <Qualifier name="telemetryLevel" value="1"/>
140:             <VariableRef name="Time" />
141:             <VariableRef name="SubS" />
142:             <VariableRef name="DeviceTemperature"/>
143:         </DataMsg>
144:     </Notification>
145: </Interface>
146:
147: </xTEDS>

```

File: sdm/app/test/DMTTests/xTEDSRegTests/GenxTEDS.sh

```
1: #!/bin/bash
2:
3: FILES=`ls *.xml`
4:
5: echo $FILES
6:
7: for file in $FILES
8: do
9:  ./xteds2str $file > ${file/xml/h}
10: done
11:
12:
```

File: sdm/app/test/DMTests/xTEDSRegTests/OOCE.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <!--Sample XML file generated by XMLSpy v2005 rel. 3 U (http://www.altova.com)-->
3:         <xTEDS          xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
version="0.1" name="OOCExTEDS"
5:   description="On-Orbit Checkout Executive">
6:
7:   <Application kind="SCL" name="OOCE"/>
8:
9:   <Interface id="1" name="OOCEInterface">
10:
11:     <Variable  format="INT08"  length="33"  kind="String"  name="ObjectiveName"
description="Identifies the ooc objective."/>
12:     <Variable format="UINT08" kind="boolean" name="ObjectiveResult" description="OOCE test
result."/>
13:     <Drange name="ObjectiveResultEnum">
14:       <Option value="0" name="FAILURE"/>
15:       <Option value="1" name="SUCCESS"/>
16:     </Drange>
17:   </Variable>
18:
19:   <Command>
20:     <CommandMsg name="InitOoceAutoSequence" description="Initializes OOCE and performs the
autonomous test sequence." id="1">
21:       </CommandMsg>
22:     </Command>
23:
24:   <Command>
25:     <CommandMsg name="InitOoce" description="Initializes OOCE prior to starting any tests."
id="2">
26:       </CommandMsg>
27:     </Command>
28:
29:   <Command>
30:     <CommandMsg name="StopOoce" description="Deactivates OOCE monitor rules, stops the test
sequence, and stops the OOCE test currently executing." id="3">
31:       </CommandMsg>
32:     </Command>
33:
```

```

34: <Command>
35:   <CommandMsg name="KillOoce" description="Deletes OOCE logs and stops sending/receiving
OOCE messages." id="4">
36:   </CommandMsg>
37: </Command>
38:
39: <Command>
40:   <CommandMsg name="StartOoceTest" description="Schedules the specified test objective."
id="5">
41:     <VariableRef name="ObjectiveName"/>
42:   </CommandMsg>
43: </Command>
44:
45: <Command>
46:   <CommandMsg name="ClearOoceStatus" description="Resets the status summary of OOCE and
all OOCE tests." id="6">
47:   </CommandMsg>
48: </Command>
49:
50: <Notification>
51:   <DataMsg name="OoceResult" description="Provides the result of executing an OOC test
objective." id="7" msgArrival="EVENT">
52:     <VariableRef name="ObjectiveName"/>
53:     <VariableRef name="ObjectiveResult"/>
54:   </DataMsg>
55: </Notification>
56:
57: </Interface>
58:
59: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/VehicleService.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS           xTEDS02.xsd"
name="Adcole_NRL_Digital_Sun_Sensor_"
4:   version="2.0">
5:   <Application name="VehicleInformationService" version="1.0" kind="VSERV" description="Broker
of all device packaging and calibration information" />
6:
7:   <Interface name="PackagingInterface" id="1" description="Used to update and request location and
orientation about components">
8:     <Variable name="Position" kind="position" length="3" format="FLOAT32" units="m"
description="Location of component frame origin in spacecraft frame.">
9:       <Qualifier name="representation" value="vector"/>
10:      <Qualifier name="frameMeasured" value="SVF"/>
11:      <Qualifier name="frameResolved" value="SVF"/>
12:    </Variable>
13:    <Variable name="Orientation" kind="attitude" length="4" format="FLOAT32"
description="Orientaiton of device frame within spacecraft frame" >
14:      <Qualifier name="representation" value="quaternion"/>
15:      <Qualifier name="frameFrom" value="SVF"/>
16:      <Qualifier name="frameTo" value="DVF"/>
17:    </Variable>
18:    <Variable name="updateType" format="UINT08" kind="mode" description="The type of update
subscription being requested">
19:      <Drange name="updateTypeEnum">
20:        <Option name="Current" value="0"/>
21:        <Option name="CurrentAndFuture" value="1"/>
22:      </Drange>
23:    </Variable>
24:    <Variable name="sensor_ID" kind="ID" format="UINT32" description="sensor ID portion of
component ID" />
25:    <Variable name="ip" kind="ipaddress" format="UINT32" description="IP address portion of
sensor ID" />
26:    <Variable name="port" kind="port" format="UINT16" description="port portion of component
ID" />
27:
28:    <Notification>
29:      <DataMsg name="PackagingInfoUpdate" id="1" msgArrival="EVENT" description="Notifies all
interested parties about a change of packaging info for a component">
30:        <VariableRef name="sensor_ID"/>
```

```

31:     <VariableRef name="ip"/>
32:     <VariableRef name="port"/>
33:     <VariableRef name="Position"/>
34:     <VariableRef name="Orientation"/>
35: </DataMsg>
36: </Notification>
37: <Command>
38:     <CommandMsg name="UpdatePackagingInfo" id="2" description="Updates packaging info of
the given component">
39:         <VariableRef name="sensor_ID"/>
40:         <VariableRef name="ip"/>
41:         <VariableRef name="port"/>
42:         <VariableRef name="Position"/>
43:         <VariableRef name="Orientation"/>
44:     </CommandMsg>
45: </Command>
46: <Command>
47:     <CommandMsg name="UpdateMyPackagingInfo" id="3" description="Updates the packaging
info of the component initiating the command">
48:         <VariableRef name="Position"/>
49:         <VariableRef name="Orientation"/>
50:     </CommandMsg>
51: </Command>
52: <Request>
53:     <CommandMsg name="GetPackagingInfo" id="4" description="Requests an update on the
packaging info for the given component">
54:         <VariableRef name="sensor_ID"/>
55:         <VariableRef name="ip"/>
56:         <VariableRef name="port"/>
57:         <VariableRef name="updateType"/>
58:     </CommandMsg>
59: <DataReplyMsg name="PackagingInfoReply" id="5" description="Reply to a request for device
packaging info">
60:     <VariableRef name="sensor_ID"/>
61:     <VariableRef name="ip"/>
62:     <VariableRef name="port"/>
63:     <VariableRef name="Position"/>
64:     <VariableRef name="Orientation"/>
65: </DataReplyMsg>
66: </Request>
67: <Command>

```

```

68:     <CommandMsg name="SubscribeToPackagingUpdates" id="6" description="Subscribes the
sender to all updates for the given component">
69:     <VariableRef name="sensor_ID"/>
70:     <VariableRef name="ip"/>
71:     <VariableRef name="port"/>
72: </CommandMsg>
73: </Command>
74: <Command>
75:     <CommandMsg name="CancelPackagingUpdates" id="7" description="Cancels the sender's
subscription to updates for the given component">
76:     <VariableRef name="sensor_ID"/>
77:     <VariableRef name="ip"/>
78:     <VariableRef name="port"/>
79: </CommandMsg>
80: </Command>
81: <Command>
82:     <CommandMsg name="CancelAllPackagingUpdates" id="8" description="Cancels all packaging
info update subscriptions held by the sender" />
83: </Command>
84: </Interface>
85:
86: </xTEDS>

```


File: sdm/app/test/DMTests/xTEDSRegTests/Thruster.h

```
1: #ifndef _MONOPROPELLANTTHRUSTER_XTEDS_H
2: #define _MONOPROPELLANTTHRUSTER_XTEDS_H
3:
4: #define _STRING_MONOPROPELLANTTHRUSTER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
   \"MonoPropellantThrusterXTEDS \"\" \
8:   "version= \"2.0 \">\" \
9:   "<Device name= \"MonopropellantThruster \" kind= \"monoThrust \" description= \"A single
   monopropellant thruster \" />\" \
10:  "<Interface name= \"ThrusterBasic \" id= \"1 \">\" \
11:  "<Qualifier name= \"headID \" value= \"0 \" />\" \
12:  "<Variable name= \"catBedReady \" format= \"UINT08 \" kind= \"valid \" units= \"0_1 \"
   description= \"Boolean byte indicating the 'ready to fire' status of the thruster \">\" \
13:  "<Drange name= \"catBedReadyEnum \">\" \
14:  "<Option name= \"notReady \" value= \"0 \" />\" \
15:  "<Option name= \"ready \" value= \"1 \" />\" \
16:  "</Drange>\" \
17:  "</Variable>\" \
18:  "<Command>\" \
19:  "<CommandMsg name= \"thrusterOnCmd \" id= \"1 \" />\" \
20:  "</Command>\" \
21:  "<Command>\" \
22:  "<CommandMsg name= \"thrusterOffCmd \" id= \"2 \" />\" \
23:  "</Command>\" \
24:  "<Request>\" \
25:  "<CommandMsg name= \"isReady \" id= \"3 \" />\" \
26:  "<DataReplyMsg name= \"thrusterStatus \" id= \"4 \">\" \
27:  "<VariableRef name= \"catBedReady \" />\" \
28:  "</DataReplyMsg>\" \
29:  "</Request>\" \
30:  "<Request>\" \
31:  "<CommandMsg name= \"prepCatBed \" id= \"4 \" />\" \
32:  "<DataReplyMsg name= \"readyToFire \" id= \"5 \" />\" \
33:  "</Request>\" \
34:  "</Interface>\" \
35:  "" \
36:  "<Interface name= \"DevPwr \" id= \"2 \">\" \
```

```

37: "<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \"/> \" \
38: "<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \"/> \" \
39: "<Qualifier name= \"CurrentHiWarning \" value= \"3.5 \" units= \"A \"/> \" \
40: "<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \"/> \" \
41: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/> \" \
42: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" \
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/> \" \
43: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \"/> \" \
44: "<Drange name= \"DevPwrStateEnum \"/> \" \
45: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" /> \" \
46: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" /> \" \
47: "</Drange>\" \
48: "</Variable>\" \
49: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \"/> \" \
50: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to \
DevPwrStateEnumeration. \">\" \
51: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" /> \" \
52: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" /> \" \
53: "</Drange>\" \
54: "</Variable>\" \
55: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length= \
\"2 \"/> \" \
56: "<Command>\" \
57: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \"/> \" \
58: "<VariableRef name= \"DevPwrStateSet \"/> \" \
59: "</CommandMsg>\" \
60: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \"/> \" \
61: "<VariableRef name= \"Time \"/> \" \
62: "<VariableRef name= \"SubS \"/> \" \
63: "<VariableRef name= \"DevPwrState \"/> \" \
64: "<VariableRef name= \"DevPwrStateSet \"/> \" \
65: "</FaultMsg>\" \
66: "</Command>\" \
67: "<Notification>\" \
68: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \"/> \" \
69: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/> \" \
70: "<VariableRef name= \"Time \"/> \" \
71: "<VariableRef name= \"SubS \"/> \" \
72: "<VariableRef name= \"DevPwrState \"/> \" \
73: "<VariableRef name= \"DevPwrStateSet \"/> \" \

```

```

74: "</DataMsg>" \
75: "</Notification>" \
76: "<Request>" \
77: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
78: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">>" \
79: "<VariableRef name= \"modePowers \"/>" \
80: "</DataReplyMsg>" \
81: "</Request>" \
82: "</Interface>" \
83: "" \
84: "<Interface name= \"CmpSOH \" id= \"3 \">>" \
85: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>" \
86: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>" \
87: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>" \
88: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>" \
89: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
90: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
91: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \"/>" \
92: "<Request>" \
93: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />" \
94: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">>" \
95: "<VariableRef name= \"Time \"/>" \
96: "<VariableRef name= \"SubS \"/>" \
97: "<VariableRef name= \"DeviceTemperature \"/>" \
98: "</DataReplyMsg>" \
99: "</Request>" \
100: "<Notification>" \
101: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">>" \
102: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
103: "<VariableRef name= \"Time \"/>" \
104: "<VariableRef name= \"SubS \"/>" \
105: "<VariableRef name= \"DeviceTemperature \"/>" \
106: "</DataMsg>" \
107: "</Notification>" \
108: "</Interface>" \
109: "</xTEDS>" \
110: ""
111:
112: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/DownlinkController.h

```
1: #ifndef _DOWNLINKCONTROLLER_XTEDS_H
2: #define _DOWNLINKCONTROLLER_XTEDS_H
3:
4: #define _STRING_DOWNLINKCONTROLLER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"DownlinkControllerXTEDS \" description= \"DownlinkController xTEDS \" version= \"1.0 \">\"
\
8:   \"\" \
9:   "<Application name= \"DownlinkController \" kind= \"CommunicationsFlightSoftware \" description=
\"Communications Subsystem Uplinked Command Controller \"/>\" \
10:   \"\" \
11:   "<Interface name= \"DownlinkControllerInterface \" id= \"1 \" description= \"Basic interface for
scheduling activities and updating their status \">\" \
12:   \"\" \
13:   "<Variable name= \"FileID \" kind= \"tbd \" format= \"UINT08 \" length= \"256 \"/>\" \
14:   \"\" \
15:   "<Command>\" \
16:   "<CommandMsg name= \"DownlinkTelemetry \" id= \"002 \" description= \"Deliver a unit of
telemetry to the downlink \"/>\" \
17:   "</Command>\" \
18:   \"\" \
19:   "<Command>\" \
20:   "<CommandMsg name= \"DownlinkFile \" id= \"003 \" description= \"Prep and begin downlinking
the referenced file \">\" \
21:   "<VariableRef name= \"FileID \"/>\" \
22:   "</CommandMsg>\" \
23:   "</Command>\" \
24:   \"\" \
25:   "<Command>\" \
26:   "<CommandMsg name= \"DownlinkSSOH \" id= \"004 \" description= \"Prep and begin downlinking
Stored State of Health \"/>\" \
27:   "</Command>\" \
28:   \"\" \
29:   "</Interface>\" \
30:   \"\" \
31:   "<Interface name= \"DownlinkControllerStatusIf \" id= \"2 \">\" \
32:   \"\" \
33:   "<Variable name= \"DownlinkControllerStatus \" kind= \"Status \" format= \"UINT08 \">\" \
```

```

34: "<Drange name= \"DownlinkControllerStatusEnum \"/>" \
35: "<Option value= \"0 \" name= \"NOT_INITIALIZED \"/>" \
36: "<Option value= \"1 \" name= \"INITIALIZING \"/>" \
37: "<Option value= \"1 \" name= \"RUNNING \"/>" \
38: "<Option value= \"2 \" name= \"TERMINATING \"/>" \
39: "</Drange>" \
40: "</Variable>" \
41: "" \
42: "<Variable name= \"DataMsgsDownlinked \"      kind= \"tbd \"      format= \"UINT32 \"/>" \
43: "<Variable name= \"DataRequestsReceived \"      kind= \"tbd \"      format= \"UINT32 \"/>" \
44: "<Variable name= \"DataMsgsInBuffer \"      kind= \"tbd \"      format= \"UINT32 \"/>" \
45: "<Variable name= \"ComponentsRegistered \"      kind= \"tbd \"      format= \"UINT16 \"/>" \
46: "<Variable name= \"FilesRequested \"      kind= \"tbd \"      format= \"UINT16 \"/>" \
47: "<Variable name= \"FilesSuccessfullyDownlinked \" kind= \"tbd \"      format= \"UINT16 \"/>" \
\
48: "<Variable name= \"FilesFailedDownlinked \"      kind= \"tbd \"      format= \"UINT16 \"/>" \
49: "<Variable name= \"CommandsAccepted \"      kind= \"tbd \"      format= \"UINT16 \"/>" \
50: "<Variable name= \"CommandsRejected \"      kind= \"tbd \"      format= \"UINT16 \"/>" \
51: "" \
52: "<Notification>" \
53: "<DataMsg name= \"DownlinkControllerStatusMsg \" id= \"001 \" msgArrival= \"EVENT \"/>" \
54: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
55: "<VariableRef name= \"DownlinkControllerStatus \"/>" \
56: "<VariableRef name= \"ComponentsRegistered \"/>" \
57: "<VariableRef name= \"DataMsgsDownlinked \"/>" \
58: "<VariableRef name= \"DataRequestsReceived \"/>" \
59: "<VariableRef name= \"DataMsgsInBuffer \"/>" \
60: "<VariableRef name= \"CommandsAccepted \"/>" \
61: "<VariableRef name= \"CommandsRejected \"/>" \
62: "<VariableRef name= \"FilesRequested \"/>" \
63: "<VariableRef name= \"FilesSuccessfullyDownlinked \"/>" \
64: "<VariableRef name= \"FilesFailedDownlinked \"/>" \
65: "</DataMsg>" \
66: "</Notification>" \
67: "" \
68: "</Interface>" \
69: "" \
70: "<Interface name= \"ICSDebugInterface \" id= \"3 \"/>" \
71: "" \
72: "<!--" \
73: "Note: DebugLevel is a bit field with the following assigned values:" \

```

```

74: "DEBUG_ENTRY_AND_EXIT = 0x01," \
75: "DEBUG_ENTRY_PARAMETERS = 0x02," \
76: "DEBUG_EXIT_PARAMETERS = 0x04," \
77: "DEBUG_LEVEL_LOW = 0x08," \
78: "DEBUG_LEVEL_MEDIUM = 0x10," \
79: "DEBUG_LEVEL_HIGH = 0x20." \
80: "The values are OR'd to determine the debug information to be logged." \
81: "-->" \
82: "" \
83: "<Variable name= \"DebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
84: "<Variable name= \"CurrentDebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
85: "" \
86: "<Variable name= \"SetDebugLevelReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \
87: "<Variable name= \"SetDebugLevelAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \
88: "<Variable name= \"SetDebugLevelSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \
89: "<Variable name= \"SetDebugLevelFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \
90: "" \
91: "<Command>" \
92: "<CommandMsg name= \"SetDebugLevel \" id= \"001 \" description= \"Set the debug log verbosity level \"/>" \
93: "<VariableRef name= \"DebugLevel \"/>" \
94: "</CommandMsg>" \
95: "</Command>" \
96: "" \
97: "<Command>" \
98: "<CommandMsg name= \"SendDebugStatus \" id= \"002 \"/>" \
99: "</Command>" \
100: "" \
101: "<Notification>" \
102: "<DataMsg name= \"DebugStatus \" id= \"003 \" msgArrival= \"EVENT \"/>" \
103: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
104: "<VariableRef name= \"CurrentDebugLevel \"/>" \
105: "<VariableRef name= \"SetDebugLevelReceived \"/>" \
106: "<VariableRef name= \"SetDebugLevelAccepted \"/>" \
107: "<VariableRef name= \"SetDebugLevelSuccess \"/>" \
108: "<VariableRef name= \"SetDebugLevelFailure \"/>" \
109: "</DataMsg>" \
110: "</Notification>" \
111: "" \
112: "</Interface>" \
113: "" \

```

```

114: "<Interface name= \"ICSTaskControlInterface \" id= \"4 \"/>" \
115: "" \
116: "<Command>" \
117: "<CommandMsg name= \"DestroyTask \" id= \"001 \"/>" \
118: "</Command>" \
119: "" \
120: "<!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->" \
121: "<!-- Other possible requests include GetPriority, and GetHeartBeatCount -->" \
122: "" \
123: "</Interface>" \
124: "" \
125: "</xTEDS>" \
126: ""
127:
128: #endif

```

```

1: #ifndef _OOCE_XTEDS_H
2: #define _OOCE_XTEDS_H
3:
4: #define _STRING_OOCE_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>" \
6: "<!--Sample XML file generated by XMLSpy v2005 rel. 3 U (http://www.altova.com)-->" \
7: "<xTEDS xmlns= \"http://www.interfacecontrol.com/SPA/xTEDS\" xmlns:xsi= \"http://www.w3.org/2001/XMLSchema-instance\" \
8: \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd\" \
9: version= \"0.1\" name= \"OOCExTEDS\" \
10: \"description= \"On-Orbit Checkout Executive\" \
11: \">\" \
12: \"<Application kind= \"SCL\" name= \"OOCE\" \
13: \">\" \
14: \"<Interface id= \"1\" name= \"OOCEInterface\" \
15: \">\" \
16: \"<Variable format= \"INT08\" length= \"33\" kind= \"String\" name= \"ObjectiveName\" \
17: description= \"Identifies the ooc objective.\" \
18: \">\" \
19: \"<Variable format= \"UINT08\" kind= \"boolean\" name= \"ObjectiveResult\" description= \"OOCE \
20: test result.\" \
21: \">\" \
22: \"<Drange name= \"ObjectiveResultEnum\" \
23: \">\" \
24: \"<Option value= \"0\" name= \"FAILURE\" \
25: \">\" \
26: \"<Option value= \"1\" name= \"SUCCESS\" \
27: \">\" \
28: \"</Drange>\" \
29: \"</Variable>\" \
30: \"</Command>\" \
31: \"<CommandMsg name= \"InitOoceAutoSequence\" description= \"Initializes OOCE and performs \
32: the autonomous test sequence.\" id= \"1\" \
33: \">\" \
34: \"</CommandMsg>\" \
35: \"</Command>\" \
36: \"<CommandMsg name= \"InitOoce\" description= \"Initializes OOCE prior to starting any tests.\" \
37: id= \"2\" \
38: \">\" \
39: \"</CommandMsg>\" \
40: \"</Command>\" \
41: \"<CommandMsg name= \"TestOoce\" description= \"Performs the test sequence.\" id= \"3\" \
42: \">\" \
43: \"</CommandMsg>\" \
44: \"</Command>\" \
45: \"</xTEDS>\" \
46: \"\" \
47: \"</Application>\" \
48: \"</Interface>\" \
49: \"</Variable>\" \
50: \"</Variable>\" \
51: \"</Drange>\" \
52: \"</Command>\" \
53: \"</CommandMsg>\" \
54: \"</CommandMsg>\" \
55: \"</xTEDS>\" \
56: \"\" \
57: \"</Application>\" \
58: \"</Interface>\" \
59: \"</Variable>\" \
60: \"</Variable>\" \
61: \"</Drange>\" \
62: \"</Command>\" \
63: \"</CommandMsg>\" \
64: \"</CommandMsg>\" \
65: \"</xTEDS>\" \
66: \"\" \
67: \"</Application>\" \
68: \"</Interface>\" \
69: \"</Variable>\" \
70: \"</Variable>\" \
71: \"</Drange>\" \
72: \"</Command>\" \
73: \"</CommandMsg>\" \
74: \"</CommandMsg>\" \
75: \"</xTEDS>\" \
76: \"\" \
77: \"</Application>\" \
78: \"</Interface>\" \
79: \"</Variable>\" \
80: \"</Variable>\" \
81: \"</Drange>\" \
82: \"</Command>\" \
83: \"</CommandMsg>\" \
84: \"</CommandMsg>\" \
85: \"</xTEDS>\" \
86: \"\" \
87: \"</Application>\" \
88: \"</Interface>\" \
89: \"</Variable>\" \
90: \"</Variable>\" \
91: \"</Drange>\" \
92: \"</Command>\" \
93: \"</CommandMsg>\" \
94: \"</CommandMsg>\" \
95: \"</xTEDS>\" \
96: \"\" \
97: \"</Application>\" \
98: \"</Interface>\" \
99: \"</Variable>\" \
100: \"</Variable>\" \
101: \"</Drange>\" \
102: \"</Command>\" \
103: \"</CommandMsg>\" \
104: \"</CommandMsg>\" \
105: \"</xTEDS>\" \
106: \"\" \
107: \"</Application>\" \
108: \"</Interface>\" \
109: \"</Variable>\" \
110: \"</Variable>\" \
111: \"</Drange>\" \
112: \"</Command>\" \
113: \"</CommandMsg>\" \
114: \"</CommandMsg>\" \
115: \"</xTEDS>\" \
116: \"\" \
117: \"</Application>\" \
118: \"</Interface>\" \
119: \"</Variable>\" \
120: \"</Variable>\" \
121: \"</Drange>\" \
122: \"</Command>\" \
123: \"</CommandMsg>\" \
124: \"</CommandMsg>\" \
125: \"</xTEDS>\" \
126: \"\" \
127: \"</Application>\" \
128: \"</Interface>\" \
129: \"</Variable>\" \
130: \"</Variable>\" \
131: \"</Drange>\" \
132: \"</Command>\" \
133: \"</CommandMsg>\" \
134: \"</CommandMsg>\" \
135: \"</xTEDS>\" \
136: \"\" \
137: \"</Application>\" \
138: \"</Interface>\" \
139: \"</Variable>\" \
140: \"</Variable>\" \
141: \"</Drange>\" \
142: \"</Command>\" \
143: \"</CommandMsg>\" \
144: \"</CommandMsg>\" \
145: \"</xTEDS>\" \
146: \"\" \
147: \"</Application>\" \
148: \"</Interface>\" \
149: \"</Variable>\" \
150: \"</Variable>\" \
151: \"</Drange>\" \
152: \"</Command>\" \
153: \"</CommandMsg>\" \
154: \"</CommandMsg>\" \
155: \"</xTEDS>\" \
156: \"\" \
157: \"</Application>\" \
158: \"</Interface>\" \
159: \"</Variable>\" \
160: \"</Variable>\" \
161: \"</Drange>\" \
162: \"</Command>\" \
163: \"</CommandMsg>\" \
164: \"</CommandMsg>\" \
165: \"</xTEDS>\" \
166: \"\" \
167: \"</Application>\" \
168: \"</Interface>\" \
169: \"</Variable>\" \
170: \"</Variable>\" \
171: \"</Drange>\" \
172: \"</Command>\" \
173: \"</CommandMsg>\" \
174: \"</CommandMsg>\" \
175: \"</xTEDS>\" \
176: \"\" \
177: \"</Application>\" \
178: \"</Interface>\" \
179: \"</Variable>\" \
180: \"</Variable>\" \
181: \"</Drange>\" \
182: \"</Command>\" \
183: \"</CommandMsg>\" \
184: \"</CommandMsg>\" \
185: \"</xTEDS>\" \
186: \"\" \
187: \"</Application>\" \
188: \"</Interface>\" \
189: \"</Variable>\" \
190: \"</Variable>\" \
191: \"</Drange>\" \
192: \"</Command>\" \
193: \"</CommandMsg>\" \
194: \"</CommandMsg>\" \
195: \"</xTEDS>\" \
196: \"\" \
197: \"</Application>\" \
198: \"</Interface>\" \
199: \"</Variable>\" \
200: \"</Variable>\" \
201: \"</Drange>\" \
202: \"</Command>\" \
203: \"</CommandMsg>\" \
204: \"</CommandMsg>\" \
205: \"</xTEDS>\" \
206: \"\" \
207: \"</Application>\" \
208: \"</Interface>\" \
209: \"</Variable>\" \
210: \"</Variable>\" \
211: \"</Drange>\" \
212: \"</Command>\" \
213: \"</CommandMsg>\" \
214: \"</CommandMsg>\" \
215: \"</xTEDS>\" \
216: \"\" \
217: \"</Application>\" \
218: \"</Interface>\" \
219: \"</Variable>\" \
220: \"</Variable>\" \
221: \"</Drange>\" \
222: \"</Command>\" \
223: \"</CommandMsg>\" \
224: \"</CommandMsg>\" \
225: \"</xTEDS>\" \
226: \"\" \
227: \"</Application>\" \
228: \"</Interface>\" \
229: \"</Variable>\" \
230: \"</Variable>\" \
231: \"</Drange>\" \
232: \"</Command>\" \
233: \"</CommandMsg>\" \
234: \"</CommandMsg>\" \
235: \"</xTEDS>\" \
236: \"\" \
237: \"</Application>\" \
238: \"</Interface>\" \
239: \"</Variable>\" \
240: \"</Variable>\" \
241: \"</Drange>\" \
242: \"</Command>\" \
243: \"</CommandMsg>\" \
244: \"</CommandMsg>\" \
245: \"</xTEDS>\" \
246: \"\" \
247: \"</Application>\" \
248: \"</Interface>\" \
249: \"</Variable>\" \
250: \"</Variable>\" \
251: \"</Drange>\" \
252: \"</Command>\" \
253: \"</CommandMsg>\" \
254: \"</CommandMsg>\" \
255: \"</xTEDS>\" \
256: \"\" \
257: \"</Application>\" \
258: \"</Interface>\" \
259: \"</Variable>\" \
260: \"</Variable>\" \
261: \"</Drange>\" \
262: \"</Command>\" \
263: \"</CommandMsg>\" \
264: \"</CommandMsg>\" \
265: \"</xTEDS>\" \
266: \"\" \
267: \"</Application>\" \
268: \"</Interface>\" \
269: \"</Variable>\" \
270: \"</Variable>\" \
271: \"</Drange>\" \
272: \"</Command>\" \
273: \"</CommandMsg>\" \
274: \"</CommandMsg>\" \
275: \"</xTEDS>\" \
276: \"\" \
277: \"</Application>\" \
278: \"</Interface>\" \
279: \"</Variable>\" \
280: \"</Variable>\" \
281: \"</Drange>\" \
282: \"</Command>\" \
283: \"</CommandMsg>\" \
284: \"</CommandMsg>\" \
285: \"</xTEDS>\" \
286: \"\" \
287: \"</Application>\" \
288
```



```

34: "<CommandMsg name= \"StopOoce \" description= \"Deactivates Ooce monitor rules, stops the test
sequence, and stops the Ooce test currently executing. \" id= \"3 \">" \
35: "</CommandMsg>" \
36: "</Command>" \
37: "" \
38: "<Command>" \
39: "<CommandMsg name= \"KillOoce \" description= \"Deletes Ooce logs and stops sending/receiving
Ooce messages. \" id= \"4 \">" \
40: "</CommandMsg>" \
41: "</Command>" \
42: "" \
43: "<Command>" \
44: "<CommandMsg name= \"StartOoceTest \" description= \"Schedules the specified test objective. \"
id= \"5 \">" \
45: "<VariableRef name= \"ObjectiveName \"/>" \
46: "</CommandMsg>" \
47: "</Command>" \
48: "" \
49: "<Command>" \
50: "<CommandMsg name= \"ClearOoceStatus \" description= \"Resets the status summary of Ooce and
all Ooce tests. \" id= \"6 \">" \
51: "</CommandMsg>" \
52: "</Command>" \
53: "" \
54: "<Notification>" \
55: "<DataMsg name= \"OoceResult \" description= \"Provides the result of executing an OOC test
objective. \" id= \"7 \" msgArrival= \"EVENT \">" \
56: "<VariableRef name= \"ObjectiveName \"/>" \
57: "<VariableRef name= \"ObjectiveResult \"/>" \
58: "</DataMsg>" \
59: "</Notification>" \
60: "" \
61: "</Interface>" \
62: "" \
63: "</xTEDS>" \
64: ""
65:
66: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/iMESA.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="iMESA_data_sheet" version="2.0">
4:   <Device   name="iMESA"   kind="ElectrostaticAnalyzer"   description="intelligent Miniature
Electrostatic Analyzer" manufacturerId="Air_Force_Academy" modelId="iMESA" />
5:
6:   <Interface name="iMESA_data" id="1">
7:     <!-- Timestamp -->
8:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
9:     <Variable   kind="SubSeconds"   name="SubS"   format="UINT32"   units="Counts"
scaleFactor=".0001" scaleUnits="Seconds"/>
10:
11:     <!-- Data Definitions -->
12:
13:     <!-- Flow Energy -->
14:     <Variable   name="FlowEnergy"   kind="Energy"   description="Plasma Flow Energy"
format="FLOAT32" length="1" units="eV"/>
15:
16:     <!-- Temperature -->
17:     <Variable   name="Temperature"   kind="Energy"   description="Plasma Flow Temperature"
format="FLOAT32" length="1" units="eV"/>
18:
19:     <!-- Temperature -->
20:     <Variable   name="Density"   kind="Density"   description="Plasma Density"   format="FLOAT32"
length="1" units="cm^-3"/>
21:
22:     <!-- Data Messages -->
23:     <Notification>
24:       <DataMsg   name="ProcessedDataStream"   description="iMESA Processed Data"
msgArrival="EVENT" id="1">
25:         <VariableRef name="SubS"/>
26:         <VariableRef name="Time"/>
27:         <VariableRef name="FlowEnergy"/>
28:         <VariableRef name="Temperature"/>
29:         <VariableRef name="Density"/>
30:       </DataMsg>
31:     </Notification>
32:
33:     <!-- Data Reply Messages -->
```

```

34:    <Request>
35:        <CommandMsg name="GetProcessedData" description="Query Processed Data" id="2" />
36:        <DataReplyMsg name="ProcessedData" description="iMESA Processed Data Once" id="3"
37:        >
38:            <VariableRef name="SubS"/>
39:            <VariableRef name="Time"/>
40:            <VariableRef name="FlowEnergy"/>
41:            <VariableRef name="Temperature"/>
42:            <VariableRef name="Density"/>
43:        </DataReplyMsg>
44:    </Request>
45: </Interface>
46: <Interface name="DevPwr" id="2">
47:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
48:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
49:     <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
50:     <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
51:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
52:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
53:     scaleFactor=".0001" scaleUnits="Seconds" />
54:     <Variable name="DevPwrState" kind="Power_State" format="UINT08">
55:         <Drange name="DevPwrStateEnum">
56:             <Option name="DevPwrOFF" value="0" description="All power to device is turned off."
57:             />
58:             <Option name="DevPwrON" value="1" description="Device may draw full power." />
59:         </Drange>
60:     </Variable>
61:     <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
62:         <Drange name="DevPwrStateEnumReference" description="This should be a reference to
63:         DevPwrStateEnumeration.">
64:             <Option name="DevPwrOFF" value="0" description="All power to device is turned off."
65:             />
66:             <Option name="DevPwrON" value="1" description="Device may draw full power." />
67:         </Drange>
68:     </Variable>
69:     <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
70:     <Command>
71:         <CommandMsg name="DevPwrSetState" id="1">
72:             <VariableRef name="DevPwrStateSet" />
73:         </CommandMsg>
74:         <FaultMsg name="DevPwrStateNotSet" id="2">

```

```

71:         <VariableRef name="Time" />
72:         <VariableRef name="SubS" />
73:         <VariableRef name="DevPwrState" />
74:         <VariableRef name="DevPwrStateSet" />
75:     </FaultMsg>
76: </Command>
77: <Notification>
78:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
79:         <Qualifier name="telemetryLevel" value="1"/>
80:         <VariableRef name="Time" />
81:         <VariableRef name="SubS" />
82:         <VariableRef name="DevPwrState" />
83:         <VariableRef name="DevPwrStateSet" />
84:     </DataMsg>
85: </Notification>
86: <Request>
87:     <CommandMsg name="getPowerInMode" id="4" />
88:     <DataReplyMsg name="powerInMode" id="5">
89:         <VariableRef name="modePowers"/>
90:     </DataReplyMsg>
91: </Request>
92: </Interface>
93:
94: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/ThreeAxisMagnetometer.h

```
1: #ifndef _THREE_AXIS_MAGNETOMETER_XTEDS_H
2: #define _THREE_AXIS_MAGNETOMETER_XTEDS_H
3:
4: #define _STRING_THREE_AXIS_MAGNETOMETER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation=  \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd  \"  name=
\"ThreeAxisMagnetometerXTEDS \"\"\" \
8:   \"version= \"2.0 \">\" \
9:   \"<Device name= \"Three_axis_magnetometer \" kind= \"mag \" description= \"3-axis output digital
magnetometer \" />\" \
10:  \"<Interface name= \"ThreeAxisMagBasic \" id= \"1 \">\" \
11:  \"<Qualifier name= \"headID \" value= \"0 \"/>\" \
12:  \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
13:  \"<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
14:  \"<Variable name= \"BVector \" kind= \"magFlux \" units= \"weber \" format= \"FLOAT32 \" length=
\"3 \" description= \"Magnetic flux observed at this device \">\" \
15:  \"<Qualifier name= \"representation \" value= \"vector \" />\" \
16:  \"<Qualifier name= \"FrameMeasured \" value= \"DVF \" />\" \
17:  \"<Qualifier name= \"FrameResolved \" value= \"DVF \" />\" \
18:  \"</Variable>\" \
19:  \"<Notification>\" \
20:  \"<DataMsg name= \"MagField \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"1 \">\" \
21:  \"<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
22:  \"<VariableRef name= \"Time \"/>\" \
23:  \"<VariableRef name= \"SubS \"/>\" \
24:  \"<VariableRef name= \"BVector \"/>\" \
25:  \"</DataMsg>\" \
26:  \"</Notification>\" \
27:  \"</Interface>\" \
28:  \"\" \
29:  \"<Interface name= \"DevPwr \" id= \"2 \">\" \
30:  \"<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \"/>\" \
31:  \"<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \"/>\" \
32:  \"<Qualifier name= \"CurrentHiWarning \" value= \"3.5 \" units= \"A \"/>\" \
33:  \"<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \"/>\" \
34:  \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
```

```

35: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \" />" \
36: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \">" \
37: "<Drange name= \"DevPwrStateEnum \">" \
38: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />"
\
39: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
40: "</Drange>" \
41: "</Variable>" \
42: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \">" \
43: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to
DevPwrStateEnumeration. \">" \
44: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />"
\
45: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
46: "</Drange>" \
47: "</Variable>" \
48: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length=
\"2 \" />" \
49: "<Command>" \
50: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \">" \
51: "<VariableRef name= \"DevPwrStateSet \" />" \
52: "</CommandMsg>" \
53: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \">" \
54: "<VariableRef name= \"Time \" />" \
55: "<VariableRef name= \"SubS \" />" \
56: "<VariableRef name= \"DevPwrState \" />" \
57: "<VariableRef name= \"DevPwrStateSet \" />" \
58: "</FaultMsg>" \
59: "</Command>" \
60: "<Notification>" \
61: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">" \
62: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
63: "<VariableRef name= \"Time \" />" \
64: "<VariableRef name= \"SubS \" />" \
65: "<VariableRef name= \"DevPwrState \" />" \
66: "<VariableRef name= \"DevPwrStateSet \" />" \
67: "</DataMsg>" \
68: "</Notification>" \
69: "<Request>" \
70: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
71: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">" \

```

```

72: "<VariableRef name= \"modePowers \"/>" \
73: "</DataReplyMsg>" \
74: "</Request>" \
75: "</Interface>" \
76: "" \
77: "<Interface name= \"CmpSOH \" id= \"3 \"/>" \
78: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>" \
79: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>" \
80: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>" \
81: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>" \
82: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
83: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
84: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \"/>" \
85: "<Request>" \
86: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \"/>" \
87: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \"/>" \
88: "<VariableRef name= \"Time \"/>" \
89: "<VariableRef name= \"SubS \"/>" \
90: "<VariableRef name= \"DeviceTemperature \"/>" \
91: "</DataReplyMsg>" \
92: "</Request>" \
93: "<Notification>" \
94: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \"/>" \
95: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
96: "<VariableRef name= \"Time \"/>" \
97: "<VariableRef name= \"SubS \"/>" \
98: "<VariableRef name= \"DeviceTemperature \"/>" \
99: "</DataMsg>" \
100: "</Notification>" \
101: "</Interface>" \
102: "</xTEDS>" \
103: ""
104:
105: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/iMESA.h

```
1: #ifndef _IMESA_XTEDS_H
2: #define _IMESA_XTEDS_H
3:
4: #define _STRING_IMESA_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
7:   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
8:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
9:   name= \"iMESA_data_sheet \" version= \"2.0 \">\" \
10:  \"<Device name= \"iMESA \" kind= \"ElectrostaticAnalyzer \" description= \"intelligent Miniature
11:  Electrostatic Analyzer \" manufacturerId= \"Air_Force_Academy \" modelId= \"iMESA \" />\" \
12:  \"<Interface name= \"iMESA_data \" id= \"1 \">\" \
13:  \"<!-- Timestamp -->\" \
14:  \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \">\" \
15:  \"<Variable kind= \"SubSeconds \" name= \"SubS \" format= \"UINT32 \" units= \"Counts \"
16:  scaleFactor= \".0001 \" scaleUnits= \"Seconds \">\" \
17:  \"<!-- Data Definitions -->\" \
18:  \"<!-- Flow Energy -->\" \
19:  \"<Variable name= \"FlowEnergy \" kind= \"Energy \" description= \"Plasma Flow Energy \" format=
20:  \"FLOAT32 \" length= \"1 \" units= \"eV \">\" \
21:  \"<!-- Temperature -->\" \
22:  \"<Variable name= \"Temperature \" kind= \"Energy \" description= \"Plasma Flow Temperature \"
23:  format= \"FLOAT32 \" length= \"1 \" units= \"eV \">\" \
24:  \"<!-- Density -->\" \
25:  \"<Variable name= \"Density \" kind= \"Density \" description= \"Plasma Density \" format=
26:  \"FLOAT32 \" length= \"1 \" units= \"cm^-3 \">\" \
27:  \"<!-- Data Messages -->\" \
28:  \"<Notification>\" \
29:  \"<DataMsg name= \"ProcessedDataStream \" description= \"iMESA Processed Data \" msgArrival=
30:  \"EVENT \" id= \"1 \">\" \
31:  \"<VariableRef name= \"SubS \">\" \
32:  \"<VariableRef name= \"Time \">\" \
33:  \"<VariableRef name= \"FlowEnergy \">\" \
34:  \"<VariableRef name= \"Temperature \">\" \
35:  \"<VariableRef name= \"Density \">\" \
```



```

34: "</DataMsg>" \
35: "</Notification>" \
36: "" \
37: "<!-- Data Reply Messages -->" \
38: "<Request>" \
39: "<CommandMsg name= \"GetProcessedData \" description= \"Query Processed Data \" id= \"2 \" />" \
40: "<DataReplyMsg name= \"ProcessedData \" description= \"iMESA Processed Data Once \" id= \"3 \">" \
41: "<VariableRef name= \"SubS \"/>" \
42: "<VariableRef name= \"Time \"/>" \
43: "<VariableRef name= \"FlowEnergy \"/>" \
44: "<VariableRef name= \"Temperature \"/>" \
45: "<VariableRef name= \"Density \"/>" \
46: "</DataReplyMsg>" \
47: "</Request>" \
48: "</Interface>" \
49: "" \
50: "<Interface name= \"DevPwr \" id= \"2 \"/>" \
51: "<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \"/>" \
52: "<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \"/>" \
53: "<Qualifier name= \"CurrentHiWarning \" value= \"3.5 \" units= \"A \"/>" \
54: "<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \"/>" \
55: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
56: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
57: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \"/>" \
58: "<Drange name= \"DevPwrStateEnum \"/>" \
59: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />" \
60: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
61: "</Drange>" \
62: "</Variable>" \
63: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \"/>" \
64: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to DevPwrStateEnumeration. \" />" \
65: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />" \
66: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
67: "</Drange>" \
68: "</Variable>" \

```

```

69: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length=
\"2 \" />\" \
70: "<Command>\" \
71: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \">\" \
72: "<VariableRef name= \"DevPwrStateSet \" />\" \
73: "</CommandMsg>\" \
74: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \">\" \
75: "<VariableRef name= \"Time \" />\" \
76: "<VariableRef name= \"SubS \" />\" \
77: "<VariableRef name= \"DevPwrState \" />\" \
78: "<VariableRef name= \"DevPwrStateSet \" />\" \
79: "</FaultMsg>\" \
80: "</Command>\" \
81: "<Notification>\" \
82: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">\" \
83: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
84: "<VariableRef name= \"Time \" />\" \
85: "<VariableRef name= \"SubS \" />\" \
86: "<VariableRef name= \"DevPwrState \" />\" \
87: "<VariableRef name= \"DevPwrStateSet \" />\" \
88: "</DataMsg>\" \
89: "</Notification>\" \
90: "<Request>\" \
91: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />\" \
92: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">\" \
93: "<VariableRef name= \"modePowers \"/>\" \
94: "</DataReplyMsg>\" \
95: "</Request>\" \
96: "</Interface>\" \
97: "" \
98: "</xTEDS>\" \
99: ""
100:
101: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/RWheelAssy.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS                                xTEDS02.xsd"
name="ReactionWheelAssemblyXxTEDS"
4:   version="2.0">
5:   <Device name="ReactionWheelAssembly" kind="rwa" description="xTeds for a 3-axis reaction
wheel assembly" />
6:
7:   <!--The assembly can be commanded as a unit, or each wheel can be commanded individually-->
8:   <!--The first interface provides the aggregate behavior-->
9:   <Interface name="RWAssyInterface" id="1">
10:     <Qualifier name="headID" value="0"/>
11:     <Variable name="torqueCommanded" kind="torque" format="FLOAT32" length="3"
units="Nm" description="Torque vector to be applied to the spacecraft">
12:       <Qualifier name="Representation" value="vector" />
13:       <Qualifier name="Frame_Measured" value="DVF" />
14:       <Qualifier name="Frame_Resolved" value="DVF" />
15:     </Variable>
16:     <Command>
17:       <!-- note: the name of a command uniquely identifies it within the dictionary -->
18:       <CommandMsg name="TorqueVectorCmd" id="1" description="A 3D torque vector
command sent to the assembly as a whole">
19:         <VariableRef name="torqueCommanded" />
20:       </CommandMsg>
21:     </Command>
22:   </Interface>
23:
24:   <!--The assembly also exposes each wheel as if it were an individual component-->
25:   <!--The first (X-Axis) wheel-->
26:   <Interface name="RWSingleInterface" id="2">
27:     <Qualifier name="headID" value="1"/>
28:     <Qualifier name="MaxMomentum" value="5.0" units="Nms" />
29:     <Qualifier name="NominalMomentum" value="2.0" units="Nms" />
30:     <Qualifier name="TorqueLossMomentum" value="1.0" units="Nms"/>
31:     <Qualifier name="MaxTorqueAtMaxSpeed" value="10.0" units="Nm" />
32:     <Qualifier name="MaxTorqueAtNominalSpeed" value="5.0" units="Nm"/>
33:     <Qualifier name="TimeConstant" value="0.1" units="s" />
34:     <Qualifier name="IdlePower" value="1.0" units="W"/>
35:     <Qualifier name="MaxPower" value="10.0" units="W"/>
```

```

36: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
37:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
38: <Variable name="opMode" kind="mode" format="UINT16" >
39:     <Drange name="enumModes" >
40:         <Option name="idle" value="0"/>
41:         <Option name="operating" value="1"/>
42:     </Drange>
43: </Variable>
44: <Variable name="dataQuality" kind="DataQuality" format="UINT08">
45:     <Drange name="DataQualityEnum">
46:         <Option name="dataBad" value="0" description="data is garbage or NaN." />
47:         <Option name="dataPoor" value="1" description="data quality is poor." />
48:         <Option name="dataGood" value="2" description="data is good." />
49:     </Drange>
50: </Variable>
51:     <Variable name="commandedTorque" kind="torque" format="FLOAT32" units="Nm"
description="Magnitude of torque requested of this device" />
52:     <Variable name="currentMomentum" kind="angularMomentum" format="FLOAT32"
units="Nms" description="The current momentum of this wheel." />
53:     <Variable name="satPercentage" kind="saturationLevel" format="FLOAT32" units="percent"
description="The current percentage of saturation of this wheel." />
54: <Command>
55:     <CommandMsg name="SingleAxisTorqueCmd" id="1">
56:         <VariableRef name="commandedTorque" />
57:     </CommandMsg>
58: </Command>
59:
60: <Notification>
61:     <DataMsg name="wheelSatLevel" msgArrival="PERIODIC" msgRate="1" id="2">
62:         <Qualifier name="telemetryLevel" value="1"/>
63:         <VariableRef name="Time" />
64:         <VariableRef name="SubS" />
65:         <VariableRef name="dataQuality" />
66:         <VariableRef name="satPercentage" />
67:     </DataMsg>
68: </Notification>
69: <Notification>
70:     <DataMsg name="MomentumCurrent" msgArrival="PERIODIC" msgRate="10" id="3">
71:         <Qualifier name="telemetryLevel" value="1"/>
72:         <VariableRef name="Time" />
73:         <VariableRef name="SubS" />

```

```

74:     <VariableRef name="dataQuality" />
75:     <VariableRef name="currentMomentum" />
76: </DataMsg>
77: </Notification>
78: <Command>
79:     <CommandMsg name="setOpMode" id="4">
80:         <VariableRef name="opMode"/>
81:     </CommandMsg>
82: </Command>
83: <Notification>
84:     <DataMsg name="opModeChanged" id="5" msgArrival="EVENT">
85:         <Qualifier name="telemetryLevel" value="1"/>
86:         <VariableRef name="opMode"/>
87:     </DataMsg>
88: </Notification>
89: </Interface>
90:
91: <!--The 2nd (Y-Axis) wheel-->
92: <Interface name="RWSingleInterface" id="3">
93:     <Qualifier name="headID" value="2"/>
94:     <Qualifier name="MaxMomentum" value="5.0" units="Nms" />
95:     <Qualifier name="NominalMomentum" value="2.0" units="Nms" />
96:     <Qualifier name="TorqueLossMomentum" value="1.0" units="Nms"/>
97:     <Qualifier name="MaxTorqueAtMaxSpeed" value="10.0" units="Nm" />
98:     <Qualifier name="MaxTorqueAtNominalSpeed" value="5.0" units="Nm"/>
99:     <Qualifier name="TimeConstant" value="0.1" units="s" />
100:    <Qualifier name="IdlePower" value="1.0" units="W"/>
101:    <Qualifier name="MaxPower" value="10.0" units="W"/>
102:    <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
103:        <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
104:    <Variable name="opMode" kind="mode" format="UINT16" >
105:        <Drange name="enumModes" >
106:            <Option name="idle" value="0"/>
107:            <Option name="operating" value="1"/>
108:        </Drange>
109:    </Variable>
110:    <Variable name="dataQuality" kind="DataQuality" format="UINT08">
111:        <Drange name="DataQualityEnum">
112:            <Option name="dataBad" value="0" description="data is garbage or NaN." />
113:            <Option name="dataPoor" value="1" description="data quality is poor." />

```

```

114:     <Option name="dataGood" value="2" description="data is good." />
115:   </Drange>
116: </Variable>
117:     <Variable name="commandedTorque" kind="torque" format="FLOAT32" units="Nm"
description="Magnitude of torque requested of this device" />
118:     <Variable name="currentMomentum" kind="angularMomentum" format="FLOAT32"
units="Nms" description="The current momentum of this wheel." />
119:     <Variable name="satPercentage" kind="saturationLevel" format="FLOAT32" units="percent"
description="The current percentage of saturation of this wheel." />
120: <Command>
121:   <CommandMsg name="SingleAxisTorqueCmd" id="1">
122:     <VariableRef name="commandedTorque" />
123:   </CommandMsg>
124: </Command>
125: <Notification>
126:   <DataMsg name="wheelSatLevel" msgArrival="PERIODIC" msgRate="1" id="2">
127:     <Qualifier name="telemetryLevel" value="1"/>
128:     <VariableRef name="Time" />
129:     <VariableRef name="SubS" />
130:     <VariableRef name="dataQuality" />
131:     <VariableRef name="satPercentage" />
132:   </DataMsg>
133: </Notification>
134: <Notification>
135:   <DataMsg name="MomentumCurrent" msgArrival="PERIODIC" msgRate="10" id="3">
136:     <Qualifier name="telemetryLevel" value="1"/>
137:     <VariableRef name="Time" />
138:     <VariableRef name="SubS" />
139:     <VariableRef name="dataQuality" />
140:     <VariableRef name="currentMomentum" />
141:   </DataMsg>
142: </Notification>
143: <Command>
144:   <CommandMsg name="setOpMode" id="4">
145:     <VariableRef name="opMode"/>
146:   </CommandMsg>
147: </Command>
148: <Notification>
149:   <DataMsg name="opModeChanged" id="5" msgArrival="EVENT">
150:     <Qualifier name="telemetryLevel" value="1"/>
151:     <VariableRef name="opMode"/>
152:   </DataMsg>

```

```

153: </Notification>
154: </Interface>
155:
156: <!--The 3nd (Z-Axis) wheel-->
157: <Interface name="RWSingleInterface" id="4">
158:   <Qualifier name="headID" value="3"/>
159:   <Qualifier name="MaxMomentum" value="5.0" units="Nms" />
160:   <Qualifier name="NominalMomentum" value="2.0" units="Nms" />
161:   <Qualifier name="TorqueLossMomentum" value="1.0" units="Nms"/>
162:   <Qualifier name="MaxTorqueAtMaxSpeed" value="10.0" units="Nm" />
163:   <Qualifier name="MaxTorqueAtNominalSpeed" value="5.0" units="Nm"/>
164:   <Qualifier name="TimeConstant" value="0.1" units="s" />
165:   <Qualifier name="IdlePower" value="1.0" units="W"/>
166:   <Qualifier name="MaxPower" value="10.0" units="W"/>
167:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
168:       <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
169:   <Variable name="opMode" kind="mode" format="UINT16" >
170:     <Drange name="enumModes" >
171:       <Option name="idle" value="0"/>
172:       <Option name="operating" value="1"/>
173:     </Drange>
174:   </Variable>
175:   <Variable name="dataQuality" kind="DataQuality" format="UINT08">
176:     <Drange name="DataQualityEnum">
177:       <Option name="dataBad" value="0" description="data is garbage or NaN." />
178:       <Option name="dataPoor" value="1" description="data quality is poor." />
179:       <Option name="dataGood" value="2" description="data is good." />
180:     </Drange>
181:   </Variable>
182:       <Variable name="commandedTorque" kind="torque" format="FLOAT32" units="Nm"
description="Magnitude of torque requested of this device" />
183:       <Variable name="currentMomentum" kind="angularMomentum" format="FLOAT32"
units="Nms" description="The current momentum of this wheel." />
184:       <Variable name="satPercentage" kind="saturationLevel" format="FLOAT32" units="percent"
description="The current percentage of saturation of this wheel." />
185:   <Command>
186:     <CommandMsg name="SingleAxisTorqueCmd" id="1">
187:       <VariableRef name="commandedTorque" />
188:     </CommandMsg>
189:   </Command>
190: </Notification>

```

```

191:     <DataMsg name="wheelSatLevel" msgArrival="PERIODIC" msgRate="1" id="2">
192:         <Qualifier name="telemetryLevel" value="1"/>
193:         <VariableRef name="Time" />
194:         <VariableRef name="SubS" />
195:         <VariableRef name="dataQuality" />
196:         <VariableRef name="satPercentage" />
197:     </DataMsg>
198: </Notification>
199: <Notification>
200:     <DataMsg name="MomentumCurrent" msgArrival="PERIODIC" msgRate="10" id="3">
201:         <Qualifier name="telemetryLevel" value="1"/>
202:         <VariableRef name="Time" />
203:         <VariableRef name="SubS" />
204:         <VariableRef name="dataQuality" />
205:         <VariableRef name="currentMomentum" />
206:     </DataMsg>
207: </Notification>
208: <Command>
209:     <CommandMsg name="setOpMode" id="4">
210:         <VariableRef name="opMode"/>
211:     </CommandMsg>
212: </Command>
213: <Notification>
214:     <DataMsg name="opModeChanged" id="5" msgArrival="EVENT">
215:         <Qualifier name="telemetryLevel" value="1"/>
216:         <VariableRef name="opMode"/>
217:     </DataMsg>
218: </Notification>
219: </Interface>
220:
221: <Interface name="DevPwr" id="5">
222:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
223:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
224:     <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
225:     <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
226:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
227:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
228:     <Variable name="DevPwrState" kind="Power_State" format="UINT08">
229:         <Drange name="DevPwrStateEnum">
230:             <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />

```



```

231:     <Option name="DevPwrON" value="1" description="Device may draw full power." />
232:   </Drange>
233: </Variable>
234: <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
235:   <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
236:     <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
237:     <Option name="DevPwrON" value="1" description="Device may draw full power." />
238:   </Drange>
239: </Variable>
240: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
241: <Command>
242:   <CommandMsg name="DevPwrSetState" id="1">
243:     <VariableRef name="DevPwrStateSet" />
244:   </CommandMsg>
245:   <FaultMsg name="DevPwrStateNotSet" id="2">
246:     <VariableRef name="Time" />
247:     <VariableRef name="SubS" />
248:     <VariableRef name="DevPwrState" />
249:     <VariableRef name="DevPwrStateSet" />
250:   </FaultMsg>
251: </Command>
252: <Notification>
253:   <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
254:     <Qualifier name="telemetryLevel" value="1"/>
255:     <VariableRef name="Time" />
256:     <VariableRef name="SubS" />
257:     <VariableRef name="DevPwrState" />
258:     <VariableRef name="DevPwrStateSet" />
259:   </DataMsg>
260: </Notification>
261: <Request>
262:   <CommandMsg name="getPowerInMode" id="4" />
263:   <DataReplyMsg name="powerInMode" id="5">
264:     <VariableRef name="modePowers"/>
265:   </DataReplyMsg>
266: </Request>
267: </Interface>
268:
269: <Interface name="CmpSOH" id="6">
270:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>

```

```

271: <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
272: <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
273: <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
274: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
275:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
276: <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
277: <Request>
278:     <CommandMsg name="GetDeviceTemperature" id="1" />
279:     <DataReplyMsg name="DeviceTempReply" id="2">
280:         <VariableRef name="Time" />
281:         <VariableRef name="SubS" />
282:         <VariableRef name="DeviceTemperature"/>
283:     </DataReplyMsg>
284: </Request>
285: <Notification>
286:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
287:         <Qualifier name="telemetryLevel" value="1"/>
288:         <VariableRef name="Time" />
289:         <VariableRef name="SubS" />
290:         <VariableRef name="DeviceTemperature"/>
291:     </DataMsg>
292: </Notification>
293: </Interface>
294: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/GPS_Full.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS           xTEDS02.xsd"
name="FullGPSxTEDS"
4:   version="2.0">
5:   <Device name="FullGPSReceiver" kind="sgr" description="A GPS receiver that calculates position,
velocity, and time internally" />
6:
7:   <Interface name="GPSInterface" id="1">
8:     <Qualifier name="headID" value="0"/>
9:     <Variable name="numVisibleSats" kind="listSize" format="INT16" description="Number of
GPS satellites visible in this observation" />
10:    <Variable name="recvTime" kind="time" format="INT32" units="s" description="Receiver
clock time since GPS epoch">
11:      <Qualifier name="timeFrame" value="GPST1" />
12:    </Variable>
13:    <Variable name="PRNs" kind="ID" format="INT16" length="28" description="list of PRN
numbers of the visible GPS satellites">
14:      <Qualifier name="representation" value="array" />
15:    </Variable>
16:    <Variable name="PRN" kind="ID" format="INT16" description="PRN of a single GPS sat
generating a NAV message" />
17:    <Variable name="Prange" kind="distance" format="FLOAT32" length="28" units="km"
description="Pseudorange measurements for each visible GPS satellite">
18:      <Qualifier name="representation" value="array" />
19:    </Variable>
20:    <Variable name="epochTime" kind="epoch" format="INT16" length="6"
description="YMDHMS definition of the current GPS epoch time">
21:      <Qualifier name="representation" value="array" />
22:    </Variable>
23:    <Variable name="nav" kind="navMessage" format="FLOAT32" length="28" description="Nav
message from this satellite: ephemeris, clock corrections">
24:      <Qualifier name="representation" value="array" />
25:    </Variable>
26:    <Variable name="gpst" kind="time" format="INT32" units="s" description="Current calculated
GPS time (UTC)">
27:      <Qualifier name="timeFrame" value="UTC" />
28:    </Variable>
29:    <Variable name="R" kind="distance" format="FLOAT32" units="km" length="3"
description="Current position of the receiver in ECEF">
```

```

30:     <Qualifier name="representation" value="vector" />
31:     <Qualifier name="frameMeasured" value="ITRF" />
32:     <Qualifier name="frameResolved" value="ITRF" />
33: </Variable>
34: <Variable name="V" kind="velocity" format="FLOAT32" units="km_s" length="3"
description="Current velocity vector of the receiver in ECEF">
35:     <Qualifier name="representation" value="vector" />
36:     <Qualifier name="frameMeasured" value="ITRF" />
37:     <Qualifier name="frameResolved" value="ITRF" />
38: </Variable>
39: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
40: <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
41: <Notification>
42:     <DataMsg name="GPSSatPrange" msgArrival="PERIODIC" msgRate="1" id="1">
43:     <Qualifier name="telemetryLevel" value="1"/>
44:         <VariableRef name="numVisibleSats" />
45:         <VariableRef name="recvTime" />
46:         <VariableRef name="PRNs" />
47:         <VariableRef name="Prange" />
48:     </DataMsg>
49: </Notification>
50: <Notification>
51:     <DataMsg name="GPSNavMessage" msgArrival="PERIODIC" msgRate="1" id="2">
52:         <VariableRef name="PRN" />
53:         <VariableRef name="epochTime" />
54:         <VariableRef name="nav" />
55:     </DataMsg>
56: </Notification>
57: <Notification>
58:     <DataMsg name="SCPosition" msgArrival="PERIODIC" msgRate="1" id="3">
59:     <Qualifier name="telemetryLevel" value="1"/>
60:     <VariableRef name="Time"/>
61:     <VariableRef name="SubS"/>
62:         <VariableRef name="gpst" />
63:         <VariableRef name="R" />
64:     </DataMsg>
65: </Notification>
66: <Notification>
67:     <DataMsg name="SCVelocity" msgArrival="PERIODIC" msgRate="1" id="4">
68:     <Qualifier name="telemetryLevel" value="1"/>

```

```

69:     <VariableRef name="Time"/>
70:     <VariableRef name="SubS"/>
71:         <VariableRef name="gpst" />
72:         <VariableRef name="V" />
73:     </DataMsg>
74: </Notification>
75: <Notification>
76:     <DataMsg name="GPSTime" msgArrival="PERIODIC" msgRate="1" id="5">
77:     <Qualifier name="telemetryLevel" value="1"/>
78:         <VariableRef name="gpst" />
79:     </DataMsg>
80: </Notification>
81: </Interface>
82:
83: <Interface name="DevPwr" id="2">
84:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
85:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
86:     <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
87:     <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
88:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
89:         <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
90:     <Variable name="DevPwrState" kind="Power_State" format="UINT08">
91:     <Drange name="DevPwrStateEnum">
92:         <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
93:         <Option name="DevPwrON" value="1" description="Device may draw full power." />
94:     </Drange>
95: </Variable>
96: <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
97:     <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
98:         <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
99:         <Option name="DevPwrON" value="1" description="Device may draw full power." />
100:     </Drange>
101: </Variable>
102: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
103: <Command>
104:     <CommandMsg name="DevPwrSetState" id="1">
105:         <VariableRef name="DevPwrStateSet" />
106:     </CommandMsg>
107:     <FaultMsg name="DevPwrStateNotSet" id="2">

```

```

108:     <VariableRef name="Time" />
109:     <VariableRef name="SubS" />
110:     <VariableRef name="DevPwrState" />
111:     <VariableRef name="DevPwrStateSet" />
112: </FaultMsg>
113: </Command>
114: <Notification>
115:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
116:         <Qualifier name="telemetryLevel" value="1"/>
117:         <VariableRef name="Time" />
118:         <VariableRef name="SubS" />
119:         <VariableRef name="DevPwrState" />
120:         <VariableRef name="DevPwrStateSet" />
121:     </DataMsg>
122: </Notification>
123: <Request>
124:     <CommandMsg name="getPowerInMode" id="4" />
125:     <DataReplyMsg name="powerInMode" id="5">
126:         <VariableRef name="modePowers"/>
127:     </DataReplyMsg>
128: </Request>
129: </Interface>
130:
131: <Interface name="CmpSOH" id="3">
132:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
133:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
134:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
135:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
136:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
137:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
138:     <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
139:     <Request>
140:         <CommandMsg name="GetDeviceTemperature" id="1" />
141:         <DataReplyMsg name="DeviceTempReply" id="2">
142:             <VariableRef name="Time" />
143:             <VariableRef name="SubS" />
144:             <VariableRef name="DeviceTemperature"/>
145:         </DataReplyMsg>
146:     </Request>
147: </Notification>

```

```
148:    <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
149:      <Qualifier name="telemetryLevel" value="1"/>
150:      <VariableRef name="Time" />
151:      <VariableRef name="SubS" />
152:      <VariableRef name="DeviceTemperature"/>
153:    </DataMsg>
154:  </Notification>
155: </Interface>
156: </xTEDS>
```

File: sdm/app/test/DMTests/xTEDSRegTests/StarCamera.h

```
1: #ifndef _TERMAHE5ASSTARTRACKER_XTEDS_H
2: #define _TERMAHE5ASSTARTRACKER_XTEDS_H
3:
4: #define _STRING_TERMAHE5ASSTARTRACKER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation=  \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
   \"StarTrackerxTEDS \"\" \
8:   "version= \"2.0 \">\" \
9:   "<Device name= \"TermaHE5ASStarTracker \" kind= \"scam \" description= \"Terma HE-5as Star
   Camera \" />\" \
10:   "" \
11:   "<Interface name= \"ScamBasic \" id= \"1 \">\" \
12:   "<Qualifier name= \"headID \" value= \"0 \" />\" \
13:   "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
14:   "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
   scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
15:   "<Variable name= \"AttitudeDev \" kind= \"attitude \" format= \"FLOAT32 \" length= \"4 \"
   description= \"Quaternion describing attitude of the device in inertial space (ECI) \" >\" \
16:   "<Qualifier name= \"representation \" value= \"quaternion \" />\" \
17:   "<Qualifier name= \"frameFrom \" value= \"ECIMOD \" />\" \
18:   "<Qualifier name= \"frameTo \" value= \"DVF \" />\" \
19:   "</Variable>\" \
20:   "<Variable name= \"AngularRate \" kind= \"attitudeRate \" format= \"FLOAT32 \" length= \"3 \"
   units= \"rad_s \" description= \"Rate of spin about each device axis \" >\" \
21:   "<Qualifier name= \"representation \" value= \"vector \" />\" \
22:   "<Qualifier name= \"frameMeasured \" value= \"DVF \" />\" \
23:   "<Qualifier name= \"frameResolved \" value= \"DVF \" />\" \
24:   "</Variable>\" \
25:   "<Notification>\" \
26:   "<DataMsg name= \"AnglRate \" msgArrival= \"PERIODIC \" msgRate= \"10 \" id= \"1 \">\" \
27:   "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
28:   "<VariableRef name= \"Time \" />\" \
29:   "<VariableRef name= \"SubS \" />\" \
30:   "<VariableRef name= \"AngularRate \" />\" \
31:   "</DataMsg>\" \
32:   "</Notification>\" \
33:   "<Notification>\" \
34:   "<DataMsg name= \"Attitude \" msgArrival= \"PERIODIC \" msgRate= \"10 \" id= \"2 \">\" \
```



```

35: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
36: "<VariableRef name= \"Time \"/>" \
37: "<VariableRef name= \"SubS \"/>" \
38: "<VariableRef name= \"AttitudeDev \" />" \
39: "</DataMsg>" \
40: "</Notification>" \
41: "</Interface>" \
42: "" \
43: "<Interface name= \"DevPwr \" id= \"2 \"/>" \
44: "<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \"/>" \
45: "<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \"/>" \
46: "<Qualifier name= \"CurrentHiWarning \" value= \"3.5 \" units= \"A \"/>" \
47: "<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \"/>" \
48: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
49: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
50: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \"/>" \
51: "<Drange name= \"DevPwrStateEnum \"/>" \
52: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \"/>" \
53: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \"/>" \
54: "</Drange>" \
55: "</Variable>" \
56: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \"/>" \
57: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to DevPwrStateEnumeration. \"/>" \
58: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \"/>" \
59: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \"/>" \
60: "</Drange>" \
61: "</Variable>" \
62: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length= \"2 \"/>" \
63: "<Command>" \
64: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \"/>" \
65: "<VariableRef name= \"DevPwrStateSet \"/>" \
66: "</CommandMsg>" \
67: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \"/>" \
68: "<VariableRef name= \"Time \"/>" \
69: "<VariableRef name= \"SubS \"/>" \
70: "<VariableRef name= \"DevPwrState \"/>" \
71: "<VariableRef name= \"DevPwrStateSet \"/>" \

```

```

72: "</FaultMsg>" \
73: "</Command>" \
74: "<Notification>" \
75: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">" \
76: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
77: "<VariableRef name= \"Time \" />" \
78: "<VariableRef name= \"SubS \" />" \
79: "<VariableRef name= \"DevPwrState \" />" \
80: "<VariableRef name= \"DevPwrStateSet \" />" \
81: "</DataMsg>" \
82: "</Notification>" \
83: "<Request>" \
84: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
85: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">" \
86: "<VariableRef name= \"modePowers \"/>" \
87: "</DataReplyMsg>" \
88: "</Request>" \
89: "</Interface>" \
90: "" \
91: "<Interface name= \"CmpSOH \" id= \"3 \">" \
92: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>" \
93: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>" \
94: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>" \
95: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>" \
96: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
97: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
98: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \" />" \
99: "<Request>" \
100: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />" \
101: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">" \
102: "<VariableRef name= \"Time \" />" \
103: "<VariableRef name= \"SubS \" />" \
104: "<VariableRef name= \"DeviceTemperature \"/>" \
105: "</DataReplyMsg>" \
106: "</Request>" \
107: "<Notification>" \
108: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
109: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
110: "<VariableRef name= \"Time \" />" \

```

```
111: "<VariableRef name= \"SubS \" />" \
112: "<VariableRef name= \"DeviceTemperature \"/>" \
113: "</DataMsg>" \
114: "</Notification>" \
115: "</Interface>" \
116: "</xTEDS>" \
117: ""
118:
119: #endif
```

File: sdm/app/test/DMTests/xTEDSRegTests/ActivityAgent.h

```
1: #ifndef _ACTIVITYAGENT_XTEDS_H
2: #define _ACTIVITYAGENT_XTEDS_H
3:
4: #define _STRING_ACTIVITYAGENT_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
7:   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
8:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
9:   name= \"ActivityAgentXTEDS\" description= \"ActivityAgent xTEDS\" version= \"2.4\">\" \
10:   \"\" \
11:   \"<Application name= \"ActivityAgent\" kind= \"AutonomyFlightSoftware\" description=
12:   \"Autonomous Tasking Executive (ATE), ActivityAgent\"/>\" \
13:   \"\" \
14:   \"<!-- Note: An ActivityAgent must implement the ActivityInterface and should implement the
15:   ActivityAgentStatusInterface. -->\" \
16:   \"\" \
17:   \"<Interface name= \"ActivityInterface\" id= \"1\"><!-- This is a template for an ActivityAgent
18:   Interface. Unique activity parameters must be added as necessary. -->\" \
19:   \"\" \
20:   \"<Variable name= \"ActivityName\" kind= \"String\" format= \"INT08\" length= \"33\"/><!-- 33-
21:   byte, null terminated string -->\" \
22:   \"<Variable name= \"ActivityId\" kind= \"ID\" format= \"UINT32\"/>\" \
23:   \"<Variable name= \"ActivityStatus\" kind= \"Status\" format= \"INT16\"/>\" \
24:   \"<Drange name= \"ActivityStatusEnum\"/>\" \
25:   \"<Option value= \"0\" name= \"SCHEDULE_FAILURE\"/><!-- The activity could not be scheduled
26:   as requested -->\" \
27:   \"<Option value= \"1\" name= \"WAITING\"/><!-- The activity has been inserted into the schedule --
28:   >\" \
29:   \"<Option value= \"2\" name= \"ENABLED\"/><!-- The activity has been sent a command to execute
30:   -->\" \
31:   \"<Option value= \"3\" name= \"TERMINATED\"/><!-- The activity has been sent a command to
32:   abort -->\" \
33:   \"<Option value= \"4\" name= \"EXECUTING\"/><!-- The activity is currently executing -->\" \
34:   \"<Option value= \"5\" name= \"DONE_FAILURE\"/><!-- The activity has completed abnormally --
35:   >\" \
36:   \"<Option value= \"6\" name= \"DONE_SUCCESS\"/><!-- The activity has completed normally -->\"
37:   \
38:   \"<Option value= \"7\" name= \"DONE_NOT_EXECUTED\"/><!-- The activity has terminated
39:   without executing -->\" \
40:   \"</Drange>\" \
41:   \"</Variable>\" \
42:   \"\" \
```

```

30: "<!-- Add Variables for the unique activity parameters -->" \
31: "" \
32: "<Command> <!-- Provides the capability to request an Activity from the ground -->" \
33: "<CommandMsg name= \"RequestActivityCmd \" id= \"001 \" description= \"Determine activity state variables and request to be scheduled \">" \
34: "<VariableRef name= \"ActivityId \"/><!-- Use 0 for on-board requests. The ActivityId will then be assigned by the ActivityManager -->" \
35: "" \
36: "<!-- Add VariableRefs for the unique activity parameters -->" \
37: "" \
38: "</CommandMsg>" \
39: "</Command>" \
40: "" \
41: "<Request>" \
42: "<CommandMsg name= \"RequestActivity \" id= \"002 \" description= \"Determine activity state variables and request to be scheduled \">" \
43: "<VariableRef name= \"ActivityId \"/><!-- Use 0 for on-board requests. The ActivityId will then be assigned by the ActivityManager -->" \
44: "" \
45: "<!-- Add VariableRefs for the unique activity parameters -->" \
46: "" \
47: "</CommandMsg>" \
48: "<DataReplyMsg name= \"RequestActivityReply \" id= \"003 \">" \
49: "<VariableRef name= \"ActivityId \"/>" \
50: "<VariableRef name= \"ActivityStatus \"/>" \
51: "</DataReplyMsg>" \
52: "</Request>" \
53: "" \
54: "<Command> <!-- Provides the capability to update an Activity from the ground -->" \
55: "<CommandMsg name= \"UpdateRequestCmd \" id= \"004 \" description= \"Update activity state variables and request a schedule update if necessary \">" \
56: "<VariableRef name= \"ActivityId \"/>" \
57: "" \
58: "<!-- Add VariableRefs for the unique activity parameters -->" \
59: "" \
60: "</CommandMsg>" \
61: "</Command>" \
62: "" \
63: "<Request>" \
64: "<CommandMsg name= \"UpdateRequest \" id= \"005 \" description= \"Update activity state variables and request a schedule update if necessary \">" \
65: "<VariableRef name= \"ActivityId \"/>" \

```

```

66: "" \
67: "<!-- Add VariableRefs for the unique activity parameters -->" \
68: "" \
69: "</CommandMsg>" \
70: "<DataReplyMsg name= \"UpdateRequestReply \" id= \"006 \">" \
71: "<VariableRef name= \"ActivityId \"/>" \
72: "<VariableRef name= \"ActivityStatus \"/>" \
73: "</DataReplyMsg>" \
74: "</Request>" \
75: "" \
76: "<Command>" \
77: "<CommandMsg name= \"Reschedule \" id= \"007 \" description= \"The activity has been removed
from the schedule and needs to be rescheduled \">" \
78: "<VariableRef name= \"ActivityId \"/>" \
79: "</CommandMsg>" \
80: "</Command>" \
81: "" \
82: "<Command>" \
83: "<CommandMsg name= \"Delete \" id= \"008 \" description= \"Delete the activity \">" \
84: "<VariableRef name= \"ActivityId \"/>" \
85: "</CommandMsg>" \
86: "</Command>" \
87: "" \
88: "<Command>" \
89: "<CommandMsg name= \"Execute \" id= \"009 \" description= \"Execute the activity \">" \
90: "<VariableRef name= \"ActivityId \"/>" \
91: "</CommandMsg>" \
92: "</Command>" \
93: "" \
94: "<Command>" \
95: "<CommandMsg name= \"Abort \" id= \"010 \" description= \"Abort execution of the activity \">" \
96: "<VariableRef name= \"ActivityId \"/>" \
97: "</CommandMsg>" \
98: "</Command>" \
99: "" \
100: "<Command>" \
101: "<CommandMsg name= \"SendActivityStatusMsg \" id= \"011 \" description= \"Send the
ActivityStatusMsg DataMsg \">" \
102: "<VariableRef name= \"ActivityId \"/>" \
103: "</CommandMsg>" \
104: "</Command>" \

```

```

105: "" \
106: "<Notification>" \
107: "<DataMsg name= \"ActivityStatusMsg\" id= \"012\" msgArrival= \"EVENT \"/>" \
108: "<Qualifier value= \"1\" name= \"telemetryLevel \"/>" \
109: "<VariableRef name= \"ActivityId \"/>" \
110: "<VariableRef name= \"ActivityName \"/>" \
111: "<VariableRef name= \"ActivityStatus \"/>" \
112: "</DataMsg>" \
113: "</Notification>" \
114: "" \
115: "</Interface>" \
116: "" \
117: "<Interface name= \"ActivityAgentStatusInterface\" id= \"2\">" \
118: "" \
119: "<Variable name= \"ActivityAgentStatus\" kind= \"Status\" format= \"INT16 \"/>" \
120: "<Drange name= \"ActivityAgentStatusEnum \"/>" \
121: "<Option value= \"0\" name= \"NOT_INITIALIZED \"/><!-- The ActivityAgent has not been
successfully initialized -->" \
122: "<Option value= \"1\" name= \"INITIALIZING \"/><!-- The ActivityAgent is in the process of
initializing -->" \
123: "<Option value= \"2\" name= \"RUNNING \"/><!-- The ActivityAgent is initialized and is running -
->" \
124: "<Option value= \"3\" name= \"TERMINATING \"/><!-- The ActivityAgent is shutting down -->"
\
125: "</Drange>" \
126: "</Variable>" \
127: "" \
128: "<Variable name= \"ActivityAgentName\" kind= \"tbd\" format= \"UINT08\" length= \"33 \"/><!--
33-byte, null terminated string -->" \
129: "" \
130: "<Variable name= \"ActivitiesCurrentlyScheduled\" kind= \"tbd\" format= \"UINT16 \"/>" \
131: "" \
132: "<Variable name= \"ActivitiesExecuted\" kind= \"tbd\" format= \"UINT16 \"/>" \
133: "<Variable name= \"ActivitiesExecutedSuccess\" kind= \"tbd\" format= \"UINT16 \"/>" \
134: "<Variable name= \"ActivitiesExecutedFailure\" kind= \"tbd\" format= \"UINT16 \"/>" \
135: "<Variable name= \"ActivitiesDeleted\" kind= \"tbd\" format= \"UINT16 \"/>" \
136: "" \
137: "<Variable name= \"RequestActivityReceived\" kind= \"tbd\" format= \"UINT16 \"/>" \
138: "<Variable name= \"RequestActivityAccepted\" kind= \"tbd\" format= \"UINT16 \"/>" \
139: "<Variable name= \"RequestActivitySuccess\" kind= \"tbd\" format= \"UINT16 \"/>" \
140: "<Variable name= \"RequestActivityFailure\" kind= \"tbd\" format= \"UINT16 \"/>" \
141: "" \

```

142: "<Variable name= \"UpdateRequestReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 143: "<Variable name= \"UpdateRequestAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 144: "<Variable name= \"UpdateRequestSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 145: "<Variable name= \"UpdateRequestFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 146: \"\" \\
 147: "<Variable name= \"ExecuteReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 148: "<Variable name= \"ExecuteAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 149: "<Variable name= \"ExecuteSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 150: "<Variable name= \"ExecuteFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 151: \"\" \\
 152: "<Variable name= \"RescheduleReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 153: "<Variable name= \"RescheduleAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 154: "<Variable name= \"RescheduleSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 155: "<Variable name= \"RescheduleFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 156: \"\" \\
 157: "<Variable name= \"DeleteReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 158: "<Variable name= \"DeleteAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 159: "<Variable name= \"DeleteSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 160: "<Variable name= \"DeleteFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 161: \"\" \\
 162: "<Command>\" \\
 163: "<CommandMsg name= \"SendActivityAgentStatusMsg \" id= \"001 \" description= \"Send the ActivityAgentStatusMsg DataMsg \"/>\" \\
 164: "</Command>\" \\
 165: \"\" \\
 166: "<Notification>\" \\
 167: "<DataMsg name= \"ActivityAgentStatusMsg \" id= \"002 \" msgArrival= \"EVENT \"/>\" \\
 168: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>\" \\
 169: "<VariableRef name= \"ActivityAgentStatus \"/>\" \\
 170: "<VariableRef name= \"ActivityAgentName \"/>\" \\
 171: "<VariableRef name= \"ActivitiesCurrentlyScheduled \"/>\" \\
 172: "<VariableRef name= \"ActivitiesExecuted \"/>\" \\
 173: "<VariableRef name= \"ActivitiesExecutedSuccess \"/>\" \\
 174: "<VariableRef name= \"ActivitiesExecutedFailure \"/>\" \\
 175: "<VariableRef name= \"ActivitiesDeleted \"/>\" \\
 176: "<VariableRef name= \"RequestActivityReceived \"/>\" \\
 177: "<VariableRef name= \"RequestActivityAccepted \"/>\" \\
 178: "<VariableRef name= \"RequestActivitySuccess \"/>\" \\
 179: "<VariableRef name= \"RequestActivityFailure \"/>\" \\
 180: "<VariableRef name= \"UpdateRequestReceived \"/>\" \\
 181: "<VariableRef name= \"UpdateRequestAccepted \"/>\" \


```

182: "<VariableRef name= \"UpdateRequestSuccess \"/>" \
183: "<VariableRef name= \"UpdateRequestFailure \"/>" \
184: "<VariableRef name= \"ExecuteReceived \"/>" \
185: "<VariableRef name= \"ExecuteAccepted \"/>" \
186: "<VariableRef name= \"ExecuteSuccess \"/>" \
187: "<VariableRef name= \"ExecuteFailure \"/>" \
188: "<VariableRef name= \"RescheduleReceived \"/>" \
189: "<VariableRef name= \"RescheduleAccepted \"/>" \
190: "<VariableRef name= \"RescheduleSuccess \"/>" \
191: "<VariableRef name= \"RescheduleFailure \"/>" \
192: "<VariableRef name= \"DeleteReceived \"/>" \
193: "<VariableRef name= \"DeleteAccepted \"/>" \
194: "<VariableRef name= \"DeleteSuccess \"/>" \
195: "<VariableRef name= \"DeleteFailure \"/>" \
196: "</DataMsg>" \
197: "</Notification>" \
198: "" \
199: "</Interface>" \
200: "" \
201: "<Interface name= \"ATEDebugInterface \" id= \"3 \">" \
202: "" \
203: "<!--" \
204: "Note: DebugLevel is a bit field with the following assigned values:" \
205: "DEBUG_ENTRY_AND_EXIT  = 0x01," \
206: "DEBUG_ENTRY_PARAMETERS = 0x02," \
207: "DEBUG_EXIT_PARAMETERS  = 0x04," \
208: "DEBUG_LEVEL_LOW       = 0x08," \
209: "DEBUG_LEVEL_MEDIUM    = 0x10," \
210: "DEBUG_LEVEL_HIGH      = 0x20." \
211: "The values are OR'd to determine the debug information to be logged." \
212: "-->" \
213: "" \
214: "<Variable name= \"DebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
215: "<Variable name= \"CurrentDebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
216: "" \
217: "<Variable name= \"SetDebugLevelReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \
218: "<Variable name= \"SetDebugLevelAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \
219: "<Variable name= \"SetDebugLevelSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \
220: "<Variable name= \"SetDebugLevelFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \
221: "" \
222: "<Command>" \

```

```

223: "<CommandMsg name= \"SetDebugLevel \" id= \"001 \" description= \"Set the debug log verbosity
level \">" \
224: "<VariableRef name= \"DebugLevel \"/>" \
225: "</CommandMsg>" \
226: "</Command>" \
227: "" \
228: "<Command>" \
229: "<CommandMsg name= \"SendDebugStatus \" id= \"002 \"/>" \
230: "</Command>" \
231: "" \
232: "<Notification>" \
233: "<DataMsg name= \"DebugStatus \" id= \"003 \" msgArrival= \"EVENT \">" \
234: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
235: "<VariableRef name= \"CurrentDebugLevel \"/>" \
236: "<VariableRef name= \"SetDebugLevelReceived \"/>" \
237: "<VariableRef name= \"SetDebugLevelAccepted \"/>" \
238: "<VariableRef name= \"SetDebugLevelSuccess \"/>" \
239: "<VariableRef name= \"SetDebugLevelFailure \"/>" \
240: "</DataMsg>" \
241: "</Notification>" \
242: "" \
243: "</Interface>" \
244: "" \
245: "<Interface name= \"TaskControlInterface \" id= \"4 \">" \
246: "" \
247: "<Command>" \
248: "<CommandMsg name= \"DestroyTask \" id= \"001 \"/>" \
249: "</Command>" \
250: "" \
251: "<!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->"
\
252: "<!-- Other possible requests include GetPriority, and GetHeartBeatCount -->" \
253: "" \
254: "</Interface>" \
255: "" \
256: "</xTEDS>" \
257: ""
258:
259: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/MagTorqueRod.h

```
1: #ifndef _BASICMAGNETICTORQUEROD_XTEDS_H
2: #define _BASICMAGNETICTORQUEROD_XTEDS_H
3:
4: #define _STRING_BASICMAGNETICTORQUEROD_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
   \"MagneticTorqueRodxTeds \"\" \
8:   \"version= \"2.0 \">\" \
9:   "<Device name= \"BasicMagneticTorqueRod \" kind= \"mtr \" description= \"Basic (on/off + polarity)
   magnetic torquer \" />\" \
10:  "<Interface name= \"MagTorquerBasic \" id= \"1 \">\" \
11:  "<Qualifier name= \"headID \" value= \"0 \" />\" \
12:  "<Variable name= \"polarity \" kind= \"polarity \" units= \"0_1 \" format= \"UINT08 \">\" \
13:  "<Drange name= \"PolarityEnum \">\" \
14:  "<Option name= \"North \" value= \"0 \" description= \"North polarity of device dipole \" />\" \
15:  "<Option name= \"South \" value= \"1 \" description= \"South polarity of device dipole \" />\" \
16:  "</Drange>\" \
17:  "</Variable>\" \
18:  "<Command>\" \
19:  "<CommandMsg name= \"MTROnCmd \" id= \"1 \">\" \
20:  "<VariableRef name= \"polarity \" />\" \
21:  "</CommandMsg>\" \
22:  "</Command>\" \
23:  "<Command>\" \
24:  "<CommandMsg name= \"MTROffCmd \" id= \"2 \" />\" \
25:  "</Command>\" \
26:  "</Interface>\" \
27:  \"\" \
28:  "<Interface name= \"DevPwr \" id= \"2 \">\" \
29:  "<Qualifier name= \"CurrentLoKeepout \" value= \"0.0 \" units= \"A \" />\" \
30:  "<Qualifier name= \"CurrentLoWarning \" value= \"0.0 \" units= \"A \" />\" \
31:  "<Qualifier name= \"CurrentHiWarning \" value= \"3.5 \" units= \"A \" />\" \
32:  "<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \" />\" \
33:  "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
34:  "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
   scaleFactor= \".0001\" scaleUnits= \"Seconds \" />\" \
35:  "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \">\" \
36:  "<Drange name= \"DevPwrStateEnum \">\" \
```

```

37: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />"
\
38: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
39: "</Drange>" \
40: "</Variable>" \
41: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \">>" \
42: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to
DevPwrStateEnumeration. \">" \
43: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />"
\
44: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
45: "</Drange>" \
46: "</Variable>" \
47: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length=
\"2 \" />" \
48: "<Command>" \
49: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \">>" \
50: "<VariableRef name= \"DevPwrStateSet \" />" \
51: "</CommandMsg>" \
52: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \">>" \
53: "<VariableRef name= \"Time \" />" \
54: "<VariableRef name= \"SubS \" />" \
55: "<VariableRef name= \"DevPwrState \" />" \
56: "<VariableRef name= \"DevPwrStateSet \" />" \
57: "</FaultMsg>" \
58: "</Command>" \
59: "<Notification>" \
60: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">>" \
61: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
62: "<VariableRef name= \"Time \" />" \
63: "<VariableRef name= \"SubS \" />" \
64: "<VariableRef name= \"DevPwrState \" />" \
65: "<VariableRef name= \"DevPwrStateSet \" />" \
66: "</DataMsg>" \
67: "</Notification>" \
68: "<Request>" \
69: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
70: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">>" \
71: "<VariableRef name= \"modePowers \" />" \
72: "</DataReplyMsg>" \
73: "</Request>" \
74: "</Interface>" \

```

```

75: "" \
76: "<Interface name= \"CmpSOH \" id= \"3 \">>\" \
77: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>\" \
78: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>\" \
79: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>\" \
80: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>\" \
81: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>\" \
82: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>\" \
83: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units=
\"degC \"/>\" \
84: "<Request>\" \
85: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \"/>\" \
86: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">>\" \
87: "<VariableRef name= \"Time \"/>\" \
88: "<VariableRef name= \"SubS \"/>\" \
89: "<VariableRef name= \"DeviceTemperature \"/>\" \
90: "</DataReplyMsg>\" \
91: "</Request>\" \
92: "<Notification>\" \
93: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">>\" \
94: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
95: "<VariableRef name= \"Time \"/>\" \
96: "<VariableRef name= \"SubS \"/>\" \
97: "<VariableRef name= \"DeviceTemperature \"/>\" \
98: "</DataMsg>\" \
99: "</Notification>\" \
100: "</Interface>\" \
101: "</xTEDS>\" \
102: ""
103:
104: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/VehicleService.h

```
1: #ifndef _VEHICLEINFORMATIONSERVICE_XTEDS_H
2: #define _VEHICLEINFORMATIONSERVICE_XTEDS_H
3:
4: #define _STRING_VEHICLEINFORMATIONSERVICE_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
\"Adcole_NRL_Digital_Sun_Sensor_ \"\" \
8:   "version= \"2.0\">\" \
9:   "<Application name= \"VehicleInformationService \" version= \"1.0\" kind= \"VSERV \" description=
\"Broker of all device packaging and calibration information \" />\" \
10:   "" \
11:   "<Interface name= \"PackagingInterface \" id= \"1 \" description= \"Used to update and request
location and orientation about components \">>\" \
12:   "<Variable name= \"Position \" kind= \"position \" length= \"3 \" format= \"FLOAT32 \" units= \"m \"
description= \"Location of component frame origin in spacecraft frame. \">>\" \
13:   "<Qualifier name= \"representation \" value= \"vector \"/>\" \
14:   "<Qualifier name= \"frameMeasured \" value= \"SVF \"/>\" \
15:   "<Qualifier name= \"frameResolved \" value= \"SVF \"/>\" \
16:   "</Variable>\" \
17:   "<Variable name= \"Orientation \" kind= \"attitude \" length= \"4 \" format= \"FLOAT32 \"
description= \"Orientaiton of device frame within spacecraft frame \" >\" \
18:   "<Qualifier name= \"representation \" value= \"quaternion \"/>\" \
19:   "<Qualifier name= \"frameFrom \" value= \"SVF \"/>\" \
20:   "<Qualifier name= \"frameTo \" value= \"DVF \"/>\" \
21:   "</Variable>\" \
22:   "<Variable name= \"updateType \" format= \"UINT08 \" kind= \"mode \" description= \"The type of
update subscription being requested \">>\" \
23:   "<Drange name= \"updateTypeEnum \">>\" \
24:   "<Option name= \"Current \" value= \"0 \"/>\" \
25:   "<Option name= \"CurrentAndFuture \" value= \"1 \"/>\" \
26:   "</Drange>\" \
27:   "</Variable>\" \
28:   "<Variable name= \"sensor_ID \" kind= \"ID \" format= \"UINT32 \" description= \"sensor ID portion
of component ID \" />\" \
29:   "<Variable name= \"ip \" kind= \"ipaddress \" format= \"UINT32 \" description= \"IP address portion
of sensor ID \" />\" \
30:   "<Variable name= \"port \" kind= \"port \" format= \"UINT16 \" description= \"port portion of
component ID \" />\" \
31:   "" \
```

```

32: "<Notification>" \
33: "<DataMsg name= \"PackagingInfoUpdate \" id= \"1 \" msgArrival= \"EVENT \" description=
  \"Notifies all interested parties about a change of packaging info for a component \">" \
34: "<VariableRef name= \"sensor_ID \"/>" \
35: "<VariableRef name= \"ip \"/>" \
36: "<VariableRef name= \"port \"/>" \
37: "<VariableRef name= \"Position \"/>" \
38: "<VariableRef name= \"Orientation \"/>" \
39: "</DataMsg>" \
40: "</Notification>" \
41: "<Command>" \
42: "<CommandMsg name= \"UpdatePackagingInfo \" id= \"2 \" description= \"Updates packaging info
  of the given component \">" \
43: "<VariableRef name= \"sensor_ID \"/>" \
44: "<VariableRef name= \"ip \"/>" \
45: "<VariableRef name= \"port \"/>" \
46: "<VariableRef name= \"Position \"/>" \
47: "<VariableRef name= \"Orientation \"/>" \
48: "</CommandMsg>" \
49: "</Command>" \
50: "<Command>" \
51: "<CommandMsg name= \"UpdateMyPackagingInfo \" id= \"3 \" description= \"Updates the
  packaging info of the component initiating the command \">" \
52: "<VariableRef name= \"Position \"/>" \
53: "<VariableRef name= \"Orientation \"/>" \
54: "</CommandMsg>" \
55: "</Command>" \
56: "<Request>" \
57: "<CommandMsg name= \"GetPackagingInfo \" id= \"4 \" description= \"Requests an update on the
  packaging info for the given component \">" \
58: "<VariableRef name= \"sensor_ID \"/>" \
59: "<VariableRef name= \"ip \"/>" \
60: "<VariableRef name= \"port \"/>" \
61: "<VariableRef name= \"updateType \"/>" \
62: "</CommandMsg>" \
63: "<DataReplyMsg name= \"PackagingInfoReply \" id= \"5 \" description= \"Reply to a request for
  device packaging info \">" \
64: "<VariableRef name= \"sensor_ID \"/>" \
65: "<VariableRef name= \"ip \"/>" \
66: "<VariableRef name= \"port \"/>" \
67: "<VariableRef name= \"Position \"/>" \
68: "<VariableRef name= \"Orientation \"/>" \

```

```

69: "</DataReplyMsg>" \
70: "</Request>" \
71: "<Command>" \
72: "<CommandMsg name= \"SubscribeToPackagingUpdates \" id= \"6 \" description= \"Subscribes the
sender to all updates for the given component \">>" \
73: "<VariableRef name= \"sensor_ID \"/>" \
74: "<VariableRef name= \"ip \"/>" \
75: "<VariableRef name= \"port \"/>" \
76: "</CommandMsg>" \
77: "</Command>" \
78: "<Command>" \
79: "<CommandMsg name= \"CancelPackagingUpdates \" id= \"7 \" description= \"Cancels the sender's
subscription to updates for the given component \">>" \
80: "<VariableRef name= \"sensor_ID \"/>" \
81: "<VariableRef name= \"ip \"/>" \
82: "<VariableRef name= \"port \"/>" \
83: "</CommandMsg>" \
84: "</Command>" \
85: "<Command>" \
86: "<CommandMsg name= \"CancelAllPackagingUpdates \" id= \"8 \" description= \"Cancels all
packaging info update subscriptions held by the sender \" />" \
87: "</Command>" \
88: "</Interface>" \
89: "" \
90: "</xTEDS>" \
91: ""
92:
93: #endif

```


File: sdm/app/test/DMTests/xTEDSRegTests/RoboHub_lean.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS                ../Schema/xTEDS02.xsd"
name="Robo_Hub_xTEDS" version="11.16">
3:
4: <Device name="RoboHub_8_Port" kind="Robust_Hub" modelId="0001" >
5: <Qualifier name="Manufacturer" value="DataDesignCorp"/>
6: <Qualifier name="Model" value="Gen1"/>
7: <Qualifier name="SerialNumber" value="12345"/>
8: </Device>
9:
10: <!-- Panel interface -->
11: <Interface id="1" name="PanelPowerInterface" description="Panel power interface">
12: <Qualifier name="NumberOfPorts" value="4"/>
13: <Qualifier name="BreakerTripCurrent" value="30.0" units="Amps"/>
14: <Variable name="PortReference" kind="PowerPortNumber" format="UINT08"/>
15: <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
16: <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32" scaleFactor=".0001"
scaleUnits="Seconds"/>
17:     <Variable name="PanelVoltage" kind="Voltage" format="INT16" scaleFactor="0.1"
scaleUnits="Volts"/>
18:     <Variable name="BreakerCurrentArray" length="4" kind="Current" format="INT16"
scaleFactor="1.0" scaleUnits="Amps">
19: <Qualifier name="Panel" value="Array_4"/>
20: </Variable>
21: <Variable name="PortPowerStateArray" length="4" kind="enumeration" format="UINT08">
22: <Drange name="PowerStateEnum">
23: <Option name="Open" value="0"/>
24: <Option name="Closed" value="1"/>
25: </Drange>
26: </Variable>
27: <Notification>
28: <DataMsg id="1" name="PowerStatus" msgArrival="PERIODIC" msgRate="1" >
29: <Qualifier name="telemetryLevel" value="1"/>
30: <VariableRef name="Time"/>
31: <VariableRef name="SubS"/>
32: <VariableRef name="PanelVoltage"/>
33: <VariableRef name="BreakerCurrentArray"/>
34: <VariableRef name="PortPowerStateArray"/>
```

```

35: </DataMsg>
36: </Notification>
37: <Command>
38: <CommandMsg id="2" name="PortPowerOn" >
39:   <VariableRef name="PortReference"/>
40: </CommandMsg>
41: </Command>
42: <Command>
43: <CommandMsg id="3" name="PortPowerOff" >
44:   <VariableRef name="PortReference"/>
45: </CommandMsg>
46: </Command>
47: </Interface>
48:
49: <!-- Hub interface -->
50: <Interface id="2" name="RoboHubInterface" description="Interface for robust hub">
51:   <Qualifier name="NumberOfPorts" value="8"/>
52:   <Qualifier name="PortTripCurrent" value="4.5" units="Amps"/>
53:   <Variable name="PortReference" kind="HubPortNumber" format="UINT08" description="Refers to
a port for events and commands"/>
54:   <Variable name="PortEnumeration" kind="HubEnumerationStatus" format="UINT08"
description="Used to indicate hub enumeration state"/>
55:   <Variable name="PPS_Status" kind="PpsStatus" format="UINT08" description="Used to indicate
PPS status"/>
56:   <Variable name="SetTripCurrentVal" kind="TripCurrent" format="INT16" scaleFactor="0.1"
defaultValue="10" scaleUnits="Amps"/>
57:   <Variable name="PortPowerState" kind="PortPowerState" format="UINT08">
58:     <Drange name="PowerStateEnum">
59:       <Option name="Open" value="0"/>
60:       <Option name="Closed" value="1"/>
61:       <Option name="HardTrip" value="2"/>
62:       <Option name="SoftTrip" value="3"/>
63:     </Drange>
64:   </Variable>
65:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
66:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32" scaleFactor=".0001"
scaleUnits="Seconds" />
67:   <Variable name="PortVoltageArray" length="8" kind="Voltage" format="INT16" scaleFactor="0.1"
scaleUnits="Volts">
68:     <Qualifier name="HubPort" value="Array_8"/>
69:   </Variable>

```

```

70: <Variable name="PortCurrentArray" length="8" kind="Current" format="INT16" scaleFactor="0.1"
scaleUnits="Amps">
71: <Qualifier name="HubPort" value="Array_8"/>
72: </Variable>
73: <Variable name="SoftCurrentLimitArray" length="8" kind="TripCurrent" format="INT16"
scaleFactor="0.1" defaultValue="10" scaleUnits="Amps">
74: <Qualifier name="HubPort" value="Array_8"/>
75: </Variable>
76: <Variable name="PortPowerStateArray" length="8" kind="boolean" format="UINT08">
77: <Qualifier name="HubPort" value="Array_8"/>
78: <Drange name="PowerStateEnumArray">
79: <Option name="Open" value="0"/>
80: <Option name="Closed" value="1"/>
81: <Option name="HardTrip" value="2"/>
82: <Option name="SoftTrip" value="3"/>
83: </Drange>
84: </Variable>
85: <Notification>
86: <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1">
87: <Qualifier name="telemetryLevel" value="1"/>
88: <VariableRef name="Time"/>
89: <VariableRef name="SubS"/>
90: <VariableRef name="PortPowerStateArray"/>
91: <VariableRef name="PortVoltageArray"/>
92: <VariableRef name="PortCurrentArray"/>
93: </DataMsg>
94: </Notification>
95: <Notification>
96: <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
97: <Qualifier name="telemetryLevel" value="1"/>
98: <VariableRef name="Time"/>
99: <VariableRef name="SubS"/>
100: <VariableRef name="PortReference"/>
101: <VariableRef name="PortPowerState"/>
102: </DataMsg>
103: </Notification>
104: <Command>
105: <CommandMsg id="3" name="ConfigureSoftTrip">
106: <VariableRef name="PortReference"/>
107: <VariableRef name="SetTripCurrentVal"/>
108: </CommandMsg>

```

```

109: </Command>
110: <Command>
111: <CommandMsg id="4" name="PortPowerOn" >
112: <VariableRef name="PortReference"/>
113: </CommandMsg>
114: </Command>
115: <Command>
116: <CommandMsg id="5" name="PortPowerOff" >
117: <VariableRef name="PortReference"/>
118: </CommandMsg>
119: </Command>
120: <Request>
121: <CommandMsg id="6" name="GetHubStatus"/>
122: <DataReplyMsg id="7" name="HubStatusReply">
123: <VariableRef name="Time"/>
124: <VariableRef name="SubS"/>
125: <VariableRef name="PortEnumeration"/>
126: <VariableRef name="PPS_Status"/>
127: </DataReplyMsg>
128: </Request>
129: <Request>
130: <CommandMsg id="8" name="GetSoftTripLimits"/>
131: <DataReplyMsg id="9" name="SoftLimitSettingsReply">
132: <VariableRef name="Time"/>
133: <VariableRef name="SubS"/>
134: <VariableRef name="SoftCurrentLimitArray"/>
135: </DataReplyMsg>
136: </Request>
137: </Interface>
138:
139: <!-- Component safety interface -->
140: <Interface name="CmpSafety" id="3">
141: <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
142: <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
143: <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
144: <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
145: <Variable kind="Time" name="Time" format="UINT32" units="Seconds"/>
146: <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds"/>
147: <Variable name="DeviceTemperature" kind="temperature" format="INT16" scaleFactor="1.0"
scaleUnits="degC"/>

```

148: <Variable name="PanelTemperatureArray" length="8" kind="temperature" format="INT16"
scaleFactor="1.0" scaleUnits="degC">
149: <Qualifier name="PanelTemperatureChannel" value="Array_8"/>
150: </Variable>
151: <Notification>
152: <DataMsg name="DeviceTemp" id="1" msgArrival="PERIODIC" msgRate="1">
153: <Qualifier name="telemetryLevel" value="1"/>
154: <VariableRef name="Time"/>
155: <VariableRef name="SubS"/>
156: <VariableRef name="DeviceTemperature"/>
157: <VariableRef name="PanelTemperatureArray"/>
158: </DataMsg>
159: </Notification>
160: </Interface>
161:
162: </xTEDS>

File: sdm/app/test/DMTTests/xTEDSRegTests/xteds2str.c

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <string.h>
4: #include <ctype.h>
5:
6: static char* ReadFile(const char* strFileName)
7: {
8:     FILE *pFile;
9:     int iSize = 8192;
10:    int iRead = 0;
11:    char *strFile;
12:
13:    pFile = fopen(strFileName, "r");
14:    if(pFile == NULL)
15:    {
16:        return NULL;
17:    }
18:
19:    strFile = (char*)malloc(iSize);
20:
21:    do
22:    {
23:        if(iRead == iSize-1)
24:        {
25:            iSize *= 2;
26:            strFile = (char*)realloc(strFile, iSize);
27:        }
28:
29:        iRead += fread( &strFile[iRead], 1, iSize-iRead-1, pFile );
30:    } while(!feof(pFile) && !ferror(pFile));
31:
32:    strFile[iRead] = '\0';
33:
34:    if(ferror(pFile))
35:    {
36:        free(strFile);
37:        fclose(pFile);
38:        return NULL;
39:    }
```

```

40:
41: fclose(pFile);
42: return strFile;
43: }
44:
45: static char* Strip(char* strIn)
46: {
47:     int iStart;
48:     int iEnd;
49:     char *strOut;
50:
51:     for(iStart=0; iStart < strlen(strIn) && !isgraph(strIn[iStart]); iStart++);
52:     for(iEnd=strlen(strIn); iEnd > iStart && !isgraph(strIn[iEnd]); iEnd--);
53:
54:     strOut = malloc(iEnd-iStart+2);
55:     memcpy(strOut, &strIn[iStart], iEnd-iStart+1);
56:     strOut[iEnd-iStart+1] = 0;
57:
58:     free(strIn);
59:
60:     return strOut;
61: }
62:
63: static char* Replace(char* strIn, char* strFrom, char* strTo)
64: {
65:     int iCnt;
66:     int iWritten;
67:
68:     char* pCur;
69:     char* pMatch;
70:     char* strOut;
71:
72:     for(iCnt = 0, pMatch=strIn;
73:         NULL != (pMatch=strstr(pMatch, strFrom));
74:         iCnt++, pMatch+=strlen(strFrom));
75:
76:     strOut = malloc(1 + strlen(strIn) + (iCnt * ( strlen(strTo) - strlen(strFrom) )));
77:
78:     pCur = strIn;
79:     pMatch = strIn;
80:     iWritten = 0;

```

```

81:
82: while(NULL != (pMatch = strstr(pMatch, strFrom)))
83: {
84:     memcpy(&strOut[iWritten], pCur, pMatch-pCur);
85:     iWritten += pMatch-pCur;
86:     pCur    += pMatch-pCur;
87:
88:     memcpy(&strOut[iWritten], strTo, strlen(strTo));
89:     iWritten += strlen(strTo);
90:
91:     pMatch += strlen(strFrom);
92:     pCur  += strlen(strFrom);
93: }
94:
95: memcpy(&strOut[iWritten], pCur, &strIn[strlen(strIn)] - pCur + 1);
96:
97: free(strIn);
98: return strOut;
99: }
100:
101: static char** Split(char* strIn)
102: {
103:     char** aStrOut;
104:     char* pCur;
105:     char* pMatch;
106:     int iCnt;
107:
108:     for(pMatch=strIn, iCnt=0; NULL != (pMatch=strchr(pMatch, '\n')); pMatch++, iCnt++);
109:     iCnt++; /* We counted new lines, so add 1 for actual lines */
110:
111:     aStrOut = malloc( (1+iCnt) * sizeof(char*) );
112:     aStrOut[iCnt] = NULL;
113:
114:     pCur = strIn;
115:     pMatch = strIn;
116:     iCnt = 0;
117:
118:     while( NULL != (pMatch=strchr(pMatch, '\n')) )
119:     {
120:         aStrOut[iCnt] = malloc(1 + pMatch - pCur);
121:         memcpy(aStrOut[iCnt], pCur, pMatch-pCur);

```



```

122:   aStrOut[iCnt][pMatch-pCur] = '\0';
123:   pMatch++;
124:   pCur += pMatch-pCur;
125:   iCnt++;
126: }
127:
128: aStrOut[iCnt] = malloc(1 + &strIn[strlen(strIn)]-pCur);
129: memcpy(aStrOut[iCnt], pCur, 1 + &strIn[strlen(strIn)]-pCur);
130:
131: return aStrOut;
132: }
133:
134: static char* ToUpper(char* strIn)
135: {
136:   char* strOut;
137:   int iCur;
138:
139:   strOut = malloc(strlen(strIn)+1);
140:   for(iCur=0; strIn[iCur] != '\0'; iCur++)
141:   {
142:     strOut[iCur] = toupper(strIn[iCur]);
143:   }
144:   strOut[iCur] = '\0';
145:
146:   free(strIn);
147:
148:   return strOut;
149: }
150:
151: static char* GetElement(const char* strElement, const char* strXml)
152: {
153:   char* pMatch = NULL;
154:   char* pEnd   = NULL;
155:   char* pStart = NULL;
156:   char* strOut = NULL;
157:
158:   pMatch = strstr(strXml, strElement);
159:   while(pMatch != NULL)
160:   {
161:     pStart = pMatch-1;
162:     while(pStart > strXml)

```

```

163:  {
164:    if( *pStart == '<' ) break;
165:    else if( isgraph(*pStart) ) break;
166:    pStart--;
167:  }
168:
169:    if( *pStart == '<' ) break;
170:    else pMatch = strstr(pMatch+1, strElement);
171:  }
172:
173: if(pMatch == NULL) return NULL;
174:
175: pEnd = strchr(pMatch, '>');
176:
177: if(NULL == pEnd) return NULL;
178:
179: pEnd++;
180:
181: strOut = malloc( pEnd-pStart + 1 );
182:
183: memcpy(strOut, pStart, pEnd-pStart);
184: strOut[pEnd-pStart] = '\0';
185:
186: return strOut;
187: }
188:
189: static char* GetAttribute(const char* strAttribute, const char* strElement)
190: {
191:  char* pMatch;
192:  char* pEnd;
193:  char* strOut;
194:
195:  pMatch = strstr(strElement, strAttribute);
196:
197:  if(NULL == pMatch) return NULL;
198:
199:  pMatch = strchr(pMatch, '=');
200:
201:  if(NULL == pMatch) return NULL;
202:
203:  pMatch = strchr(pMatch, '\"');

```

```

204:
205: if(NULL == pMatch) return NULL;
206: pMatch++;
207:
208: pEnd = strchr(pMatch, '\\');
209:
210: if(NULL == pEnd) return NULL;
211:
212: strOut = malloc(pEnd - pMatch + 1);
213: memcpy(strOut, pMatch, pEnd-pMatch);
214: strOut[pEnd-pMatch] = '\\0';
215:
216: return strOut;
217: }
218:
219: static char* GetName(const char* strXteds)
220: {
221:     char* strElement;
222:     char* strAttribute;
223:
224:     strElement = GetElement("Device", strXteds);
225:     if( NULL == strElement )
226:     {
227:         strElement = GetElement("Application", strXteds);
228:     }
229:
230:     if(NULL == strElement) return NULL;
231:
232:     strAttribute = GetAttribute("name", strElement);
233:
234:     free(strElement);
235:     return strAttribute;
236: }
237:
238: static int Convert(char* strInFile)
239: {
240:     char* strXteds;
241:     char** aSplitXteds;
242:     char* strName;
243:
244:     int iCurLine;

```

```

245:
246: strXteds = ReadFile(strInFile);
247: if(NULL == strXteds)
248: {
249:     fprintf(stderr, "Failed to read %s \n", strInFile);
250:     return 1;
251: }
252:
253: strName = GetName(strXteds);
254: if(NULL == strName)
255: {
256:     fprintf(stderr, "Could not find Application or Device name in file %s \n", strInFile);
257:     free(strXteds);
258:     return 1;
259: }
260:
261: strName = Strip(strName);
262: strName = Replace(strName, " ", "_");
263: strName = ToUpper(strName);
264:
265: strXteds = Strip(strXteds);
266: strXteds = Replace(strXteds, "\r\n", "\n");
267: strXteds = Replace(strXteds, "\r", "\n");
268: strXteds = Replace(strXteds, "\"", "\\\"");
269:
270: aSplitXteds = Split(strXteds);
271:
272: printf("#ifndef _%s_XTEDS_H \n", strName);
273: printf("#define _%s_XTEDS_H \n", strName);
274: printf(" \n");
275: printf("#define _STRING_%s_XTEDS \\ \n", strName);
276: for(iCurLine=0; aSplitXteds[iCurLine] != NULL; iCurLine++)
277: {
278:     aSplitXteds[iCurLine] = Strip(aSplitXteds[iCurLine]);
279:     printf(" \"%s \" \\ \n", aSplitXteds[iCurLine]);
280:     free(aSplitXteds[iCurLine]);
281: }
282: free(aSplitXteds);
283:
284: printf(" \" \" \n");
285:

```

```
286: printf("\n");
287: printf("#endif\n");
288:
289: free(strName);
290: free(strXteds);
291:
292: return 0;
293: }
294:
295:
296: int main(int argc, char** argv)
297: {
298:     if(argc < 2)
299:     {
300:         printf("Usage: %s infile\n", argv[0]);
301:         return 0;
302:     }
303:
304:     return Convert(argv[1]);
305: }
```

File: sdm/app/test/DMTests/xTEDSRegTests/RoboHub.h

```
1: #ifndef _ROBOHUB_8_PORT_XTEDS_H
2: #define _ROBOHUB_8_PORT_XTEDS_H
3:
4: #define _STRING_ROBOHUB_8_PORT_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"Robo_Hub_xTEDS \" version= \"11.16 \">\" \
8:   "" \
9: "<Device name= \"RobustHub_8_Port \" kind= \"Robust_Hub \" modelId= \"0001 \">\" \
10: "<Qualifier name= \"Manufacturer \" value= \"DataDesignCorp \"/>\" \
11: "<Qualifier name= \"Model \" value= \"Gen1 \"/>\" \
12: "<Qualifier name= \"SerialNumber \" value= \"12345 \"/>\" \
13: "</Device>\" \
14: "" \
15: "<Interface id= \"1 \" name= \"InterPanelPortInterface \" description= \"Interface for one of the inter-
panel ports \">\" \
16: "<Qualifier name= \"PortID \" value= \"A \"/>\" \
17: "<Qualifier name= \"BreakerTripCurrent \" value= \"30.0 \" units= \"Amps \"/>\" \
18: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
19: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
20: "<Variable name= \"BreakerCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps
\"/>\" \
21: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \">\" \
22: "<Drange name= \"PowerStateEnum \">\" \
23: "<Option name= \"Off \" value= \"0 \"/>\" \
24: "<Option name= \"On \" value= \"1 \"/>\" \
25: "<Option name= \"Tripped \" value= \"2 \"/>\" \
26: "</Drange>\" \
27: "</Variable>\" \
28: "<Notification>\" \
29: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">\" \
30: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
31: "<VariableRef name= \"Time \" />\" \
32: "<VariableRef name= \"SubS \" />\" \
33: "<VariableRef name= \"BreakerCurrent \"/>\" \
34: "</DataMsg>\" \
35: "</Notification>\" \
```

```

36: "<Notification>" \
37: "<DataMsg id= \"2\" name= \"PortTripped\" msgArrival= \"EVENT \">" \
38: "<Qualifier name= \"telemetryLevel\" value= \"1\"/>" \
39: "<VariableRef name= \"Time\" />" \
40: "<VariableRef name= \"SubS\" />" \
41: "<VariableRef name= \"PortPowerState\" />" \
42: "</DataMsg>" \
43: "</Notification>" \
44: "<Command>" \
45: "<CommandMsg id= \"2\" name= \"PortPowerOn\" />" \
46: "</Command>" \
47: "<Command>" \
48: "<CommandMsg id= \"3\" name= \"PortPowerOff\" />" \
49: "</Command>" \
50: "</Interface>" \
51: "" \
52: "<Interface id= \"2\" name= \"InterPanelPortInterface\" description= \"Interface for one of the inter-
panel ports \">" \
53: "<Qualifier name= \"portID\" value= \"B\"/>" \
54: "<Qualifier name= \"BreakerTripCurrent\" value= \"30.0\" units= \"Amps\"/>" \
55: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />" \
56: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\"
scaleFactor= \".0001\" scaleUnits= \"Seconds\" />" \
57: "<Variable name= \"BreakerCurrent\" kind= \"Current\" format= \"FLOAT32\" scaleUnits= \"Amps
\"/>" \
58: "<Variable name= \"PortPowerState\" kind= \"boolean\" format= \"UINT08\"/>" \
59: "<Drange name= \"PowerStateEnum\"/>" \
60: "<Option name= \"Off\" value= \"0\"/>" \
61: "<Option name= \"On\" value= \"1\"/>" \
62: "<Option name= \"Tripped\" value= \"2\"/>" \
63: "</Drange>" \
64: "</Variable>" \
65: "<Notification>" \
66: "<DataMsg id= \"1\" name= \"PortStatus\" msgArrival= \"PERIODIC\" msgRate= \"1\"/>" \
67: "<Qualifier name= \"telemetryLevel\" value= \"1\"/>" \
68: "<VariableRef name= \"Time\" />" \
69: "<VariableRef name= \"SubS\" />" \
70: "<VariableRef name= \"BreakerCurrent\" />" \
71: "</DataMsg>" \
72: "</Notification>" \
73: "<Notification>" \
74: "<DataMsg id= \"2\" name= \"PortTripped\" msgArrival= \"EVENT \">" \

```

```

75: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
76: "<VariableRef name= \"Time \" />" \
77: "<VariableRef name= \"SubS \" />" \
78: "<VariableRef name= \"PortPowerState \"/>" \
79: "</DataMsg>" \
80: "</Notification>" \
81: "<Command>" \
82: "<CommandMsg id= \"2 \" name= \"PortPowerOn \" />" \
83: "</Command>" \
84: "<Command>" \
85: "<CommandMsg id= \"3 \" name= \"PortPowerOff \" />" \
86: "</Command>" \
87: "</Interface>" \
88: "" \
89: "<Interface id= \"3 \" name= \"InterPanelPortInterface \" description= \"Interface for one of the inter-
panel ports \"/>" \
90: "<Qualifier name= \"PortID \" value= \"C \"/>" \
91: "<Qualifier name= \"BreakerTripCurrent \" value= \"30.0 \" units= \"Amps \"/>" \
92: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
93: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
94: "<Variable name= \"BreakerCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps
\" />" \
95: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \"/>" \
96: "<Drange name= \"PowerStateEnum \"/>" \
97: "<Option name= \"Off \" value= \"0 \"/>" \
98: "<Option name= \"On \" value= \"1 \"/>" \
99: "<Option name= \"Tripped \" value= \"2 \"/>" \
100: "</Drange>" \
101: "</Variable>" \
102: "<Notification>" \
103: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
104: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
105: "<VariableRef name= \"Time \" />" \
106: "<VariableRef name= \"SubS \" />" \
107: "<VariableRef name= \"BreakerCurrent \"/>" \
108: "</DataMsg>" \
109: "</Notification>" \
110: "<Notification>" \
111: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \"/>" \
112: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
113: "<VariableRef name= \"Time \" />" \

```



```

114: "<VariableRef name= \"SubS \" />" \
115: "<VariableRef name= \"PortPowerState \" />" \
116: "</DataMsg>" \
117: "</Notification>" \
118: "<Command>" \
119: "<CommandMsg id= \"2 \" name= \"PortPowerOn \" />" \
120: "</Command>" \
121: "<Command>" \
122: "<CommandMsg id= \"3 \" name= \"PortPowerOff \" />" \
123: "</Command>" \
124: "</Interface>" \
125: "" \
126: "<Interface id= \"4 \" name= \"InterPanelPortInterface \" description= \"Interface for one of the inter-
panel ports \">" \
127: "<Qualifier name= \"PortID \" value= \"D \" />" \
128: "<Qualifier name= \"BreakerTripCurrent \" value= \"30.0 \" units= \"Amps \" />" \
129: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
130: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
131: "<Variable name= \"BreakerCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits=
\"Amps \" />" \
132: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \">" \
133: "<Drange name= \"PowerStateEnum \">" \
134: "<Option name= \"Off \" value= \"0 \" />" \
135: "<Option name= \"On \" value= \"1 \" />" \
136: "<Option name= \"Tripped \" value= \"2 \" />" \
137: "</Drange>" \
138: "</Variable>" \
139: "<Notification>" \
140: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
141: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
142: "<VariableRef name= \"Time \" />" \
143: "<VariableRef name= \"SubS \" />" \
144: "<VariableRef name= \"BreakerCurrent \" />" \
145: "</DataMsg>" \
146: "</Notification>" \
147: "<Notification>" \
148: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \">" \
149: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
150: "<VariableRef name= \"Time \" />" \
151: "<VariableRef name= \"SubS \" />" \
152: "<VariableRef name= \"PortPowerState \" />" \

```

```

153: "</DataMsg>" \
154: "</Notification>" \
155: "<Command>" \
156: "<CommandMsg id= \"2\" name= \"PortPowerOn\" />" \
157: "</Command>" \
158: "<Command>" \
159: "<CommandMsg id= \"3\" name= \"PortPowerOff\" />" \
160: "</Command>" \
161: "</Interface>" \
162: "" \
163: "<Interface id= \"5\" name= \"ASIMPortInterface\" description= \"Interface for a single SPA-U
port on the hub.\">" \
164: "<Qualifier name= \"PortID\" value= \"0\" />" \
165: "<Qualifier name= \"BreakerTripCurrent\" value= \"4.5\" units= \"Amps\" />" \
166: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />" \
167: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\"
scaleFactor= \".0001\" scaleUnits= \"Seconds\" />" \
168: "<Variable name= \"PortVoltage\" kind= \"Voltage\" format= \"FLOAT32\" scaleUnits= \"Volts
\" />" \
169: "<Variable name= \"PortCurrent\" kind= \"Current\" format= \"FLOAT32\" scaleUnits= \"Amps
\" />" \
170: "<Variable name= \"SoftCurrentLimit\" kind= \"TripCurrent\" format= \"FLOAT32\"
defaultValue= \"1.0\" scaleUnits= \"Amps\" />" \
171: "<Variable name= \"PortPowerState\" kind= \"boolean\" format= \"UINT08\">" \
172: "<Drange name= \"PowerStateEnum\">" \
173: "<Option name= \"Off\" value= \"0\" />" \
174: "<Option name= \"On\" value= \"1\" />" \
175: "<Option name= \"Tripped\" value= \"2\" />" \
176: "</Drange>" \
177: "</Variable>" \
178: "<Notification>" \
179: "<DataMsg id= \"1\" name= \"PortStatus\" msgArrival= \"PERIODIC\" msgRate= \"1\">" \
180: "<Qualifier name= \"telemetryLevel\" value= \"1\" />" \
181: "<VariableRef name= \"Time\" />" \
182: "<VariableRef name= \"SubS\" />" \
183: "<VariableRef name= \"PortPowerState\" />" \
184: "<VariableRef name= \"PortVoltage\" />" \
185: "<VariableRef name= \"PortCurrent\" />" \
186: "</DataMsg>" \
187: "</Notification>" \
188: "<Notification>" \
189: "<DataMsg id= \"2\" name= \"PortTripped\" msgArrival= \"EVENT\">" \

```

```

190: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
191: "<VariableRef name= \"Time \" />" \
192: "<VariableRef name= \"SubS \" />" \
193: "<VariableRef name= \"PortPowerState \"/>" \
194: "</DataMsg>" \
195: "</Notification>" \
196: "<Command>" \
197: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \"/>" \
198: "<VariableRef name= \"SoftCurrentLimit \"/>" \
199: "</CommandMsg>" \
200: "</Command>" \
201: "<Command>" \
202: "<CommandMsg id= \"4 \" name= \"PortPowerOn \" />" \
203: "</Command>" \
204: "<Command>" \
205: "<CommandMsg id= \"5 \" name= \"PortPowerOff \" />" \
206: "</Command>" \
207: "</Interface>" \
208: "" \
209: "" \
210: "<Interface id= \"6 \" name= \"ASIMPortInterface \" description= \"Interface for a single SPA-U
port on the hub. \"/>" \
211: "<Qualifier name= \"PortID \" value= \"1 \"/>" \
212: "<Qualifier name= \"BreakerTripCurrent \" value= \"4.5 \" units= \"Amps \"/>" \
213: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
214: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
215: "<Variable name= \"PortVoltage \" kind= \"Voltage \" format= \"FLOAT32 \" scaleUnits= \"Volts
\" />" \
216: "<Variable name= \"PortCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps
\" />" \
217: "<Variable name= \"SoftCurrentLimit \" kind= \"TripCurrent \" format= \"FLOAT32 \"
defaultValue= \"1.0 \" scaleUnits= \"Amps \"/>" \
218: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \"/>" \
219: "<Drange name= \"PowerStateEnum \"/>" \
220: "<Option name= \"Off \" value= \"0 \"/>" \
221: "<Option name= \"On \" value= \"1 \"/>" \
222: "<Option name= \"Tripped \" value= \"2 \"/>" \
223: "</Drange>" \
224: "</Variable>" \
225: "<Notification>" \
226: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \

```

227: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \\
 228: "<VariableRef name= \"Time \" />\" \\
 229: "<VariableRef name= \"SubS \" />\" \\
 230: "<VariableRef name= \"PortPowerState \" />\" \\
 231: "<VariableRef name= \"PortVoltage \"/>\" \\
 232: "<VariableRef name= \"PortCurrent \"/>\" \\
 233: "</DataMsg>\" \\
 234: "</Notification>\" \\
 235: "<Notification>\" \\
 236: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \">\" \\
 237: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \\
 238: "<VariableRef name= \"Time \" />\" \\
 239: "<VariableRef name= \"SubS \" />\" \\
 240: "<VariableRef name= \"PortPowerState \"/>\" \\
 241: "</DataMsg>\" \\
 242: "</Notification>\" \\
 243: "<Command>\" \\
 244: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \">\" \\
 245: "<VariableRef name= \"SoftCurrentLimit \"/>\" \\
 246: "</CommandMsg>\" \\
 247: "</Command>\" \\
 248: "<Command>\" \\
 249: "<CommandMsg id= \"4 \" name= \"PortPowerOn \" />\" \\
 250: "</Command>\" \\
 251: "<Command>\" \\
 252: "<CommandMsg id= \"5 \" name= \"PortPowerOff \" />\" \\
 253: "</Command>\" \\
 254: "</Interface>\" \\
 255: "" \\
 256: "" \\
 257: "<Interface id= \"7 \" name= \"ASIMPortInterface \" description= \"Interface for a single SPA-U port on the hub. \">\" \\
 258: "<Qualifier name= \"PortID \" value= \"2 \"/>\" \\
 259: "<Qualifier name= \"BreakerTripCurrent \" value= \"4.5 \" units= \"Amps \"/>\" \\
 260: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \\
 261: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \\
 262: "<Variable name= \"PortVoltage \" kind= \"Voltage \" format= \"FLOAT32 \" scaleUnits= \"Volts \"/>\" \\
 263: "<Variable name= \"PortCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps \"/>\" \

264: "<Variable name= \"SoftCurrentLimit \" kind= \"TripCurrent \" format= \"FLOAT32 \"
 defaultValue= \"1.0 \" scaleUnits= \"Amps \"/>\" \

265: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \"/>\" \

266: "<Drange name= \"PowerStateEnum \"/>\" \

267: "<Option name= \"Off \" value= \"0 \"/>\" \

268: "<Option name= \"On \" value= \"1 \"/>\" \

269: "<Option name= \"Tripped \" value= \"2 \"/>\" \

270: "</Drange>\" \

271: "</Variable>\" \

272: "<Notification>\" \

273: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">\" \

274: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \

275: "<VariableRef name= \"Time \" />\" \

276: "<VariableRef name= \"SubS \" />\" \

277: "<VariableRef name= \"PortPowerState \" />\" \

278: "<VariableRef name= \"PortVoltage \"/>\" \

279: "<VariableRef name= \"PortCurrent \"/>\" \

280: "</DataMsg>\" \

281: "</Notification>\" \

282: "<Notification>\" \

283: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \"/>\" \

284: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \

285: "<VariableRef name= \"Time \" />\" \

286: "<VariableRef name= \"SubS \" />\" \

287: "<VariableRef name= \"PortPowerState \"/>\" \

288: "</DataMsg>\" \

289: "</Notification>\" \

290: "<Command>\" \

291: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \"/>\" \

292: "<VariableRef name= \"SoftCurrentLimit \"/>\" \

293: "</CommandMsg>\" \

294: "</Command>\" \

295: "<Command>\" \

296: "<CommandMsg id= \"4 \" name= \"PortPowerOn \" />\" \

297: "</Command>\" \

298: "<Command>\" \

299: "<CommandMsg id= \"5 \" name= \"PortPowerOff \" />\" \

300: "</Command>\" \

301: "</Interface>\" \

302: "" \

303: "" \

```

304: "<Interface id= \"8 \" name= \"ASIMPortInterface \" description= \"Interface for a single SPA-U
port on the hub. \"/>" \
305: "<Qualifier name= \"PortID \" value= \"3 \"/>" \
306: "<Qualifier name= \"BreakerTripCurrent \" value= \"4.5 \" units= \"Amps \"/>" \
307: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
308: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
309: "<Variable name= \"PortVoltage \" kind= \"Voltage \" format= \"FLOAT32 \" scaleUnits= \"Volts
\"/>" \
310: "<Variable name= \"PortCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps
\"/>" \
311: "<Variable name= \"SoftCurrentLimit \" kind= \"TripCurrent \" format= \"FLOAT32 \"
defaultValue= \"1.0 \" scaleUnits= \"Amps \"/>" \
312: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \"/>" \
313: "<Drange name= \"PowerStateEnum \"/>" \
314: "<Option name= \"Off \" value= \"0 \"/>" \
315: "<Option name= \"On \" value= \"1 \"/>" \
316: "<Option name= \"Tripped \" value= \"2 \"/>" \
317: "</Drange>" \
318: "</Variable>" \
319: "<Notification>" \
320: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
321: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
322: "<VariableRef name= \"Time \"/>" \
323: "<VariableRef name= \"SubS \"/>" \
324: "<VariableRef name= \"PortPowerState \"/>" \
325: "<VariableRef name= \"PortVoltage \"/>" \
326: "<VariableRef name= \"PortCurrent \"/>" \
327: "</DataMsg>" \
328: "</Notification>" \
329: "<Notification>" \
330: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \"/>" \
331: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
332: "<VariableRef name= \"Time \"/>" \
333: "<VariableRef name= \"SubS \"/>" \
334: "<VariableRef name= \"PortPowerState \"/>" \
335: "</DataMsg>" \
336: "</Notification>" \
337: "<Command>" \
338: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \"/>" \
339: "<VariableRef name= \"SoftCurrentLimit \"/>" \
340: "</CommandMsg>" \

```

```

341: "</Command>" \
342: "<Command>" \
343: "<CommandMsg id= \"4\" name= \"PortPowerOn\" />" \
344: "</Command>" \
345: "<Command>" \
346: "<CommandMsg id= \"5\" name= \"PortPowerOff\" />" \
347: "</Command>" \
348: "</Interface>" \
349: "" \
350: "" \
351: "<Interface id= \"9\" name= \"ASIMPortInterface\" description= \"Interface for a single SPA-U
port on the hub.\" />" \
352: "<Qualifier name= \"PortID\" value= \"4\" />" \
353: "<Qualifier name= \"BreakerTripCurrent\" value= \"4.5\" units= \"Amps\" />" \
354: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />" \
355: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\"
scaleFactor= \".0001\" scaleUnits= \"Seconds\" />" \
356: "<Variable name= \"PortVoltage\" kind= \"Voltage\" format= \"FLOAT32\" scaleUnits= \"Volts
\" />" \
357: "<Variable name= \"PortCurrent\" kind= \"Current\" format= \"FLOAT32\" scaleUnits= \"Amps
\" />" \
358: "<Variable name= \"SoftCurrentLimit\" kind= \"TripCurrent\" format= \"FLOAT32\"
defaultValue= \"1.0\" scaleUnits= \"Amps\" />" \
359: "<Variable name= \"PortPowerState\" kind= \"boolean\" format= \"UINT08\" />" \
360: "<Drange name= \"PowerStateEnum\" />" \
361: "<Option name= \"Off\" value= \"0\" />" \
362: "<Option name= \"On\" value= \"1\" />" \
363: "<Option name= \"Tripped\" value= \"2\" />" \
364: "</Drange>" \
365: "</Variable>" \
366: "<Notification>" \
367: "<DataMsg id= \"1\" name= \"PortStatus\" msgArrival= \"PERIODIC\" msgRate= \"1\" />" \
368: "<Qualifier name= \"telemetryLevel\" value= \"1\" />" \
369: "<VariableRef name= \"Time\" />" \
370: "<VariableRef name= \"SubS\" />" \
371: "<VariableRef name= \"PortPowerState\" />" \
372: "<VariableRef name= \"PortVoltage\" />" \
373: "<VariableRef name= \"PortCurrent\" />" \
374: "</DataMsg>" \
375: "</Notification>" \
376: "<Notification>" \
377: "<DataMsg id= \"2\" name= \"PortTripped\" msgArrival= \"EVENT\" />" \

```

```

378: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
379: "<VariableRef name= \"Time \" />" \
380: "<VariableRef name= \"SubS \" />" \
381: "<VariableRef name= \"PortPowerState \"/>" \
382: "</DataMsg>" \
383: "</Notification>" \
384: "<Command>" \
385: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \"/>" \
386: "<VariableRef name= \"SoftCurrentLimit \"/>" \
387: "</CommandMsg>" \
388: "</Command>" \
389: "<Command>" \
390: "<CommandMsg id= \"4 \" name= \"PortPowerOn \" />" \
391: "</Command>" \
392: "<Command>" \
393: "<CommandMsg id= \"5 \" name= \"PortPowerOff \" />" \
394: "</Command>" \
395: "</Interface>" \
396: "" \
397: "" \
398: "<Interface id= \"10 \" name= \"ASIMPortInterface \" description= \"Interface for a single SPA-U
port on the hub. \">" \
399: "<Qualifier name= \"PortID \" value= \"5 \"/>" \
400: "<Qualifier name= \"BreakerTripCurrent \" value= \"4.5 \" units= \"Amps \"/>" \
401: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
402: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
403: "<Variable name= \"PortVoltage \" kind= \"Voltage \" format= \"FLOAT32 \" scaleUnits= \"Volts
\"/>" \
404: "<Variable name= \"PortCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps
\"/>" \
405: "<Variable name= \"SoftCurrentLimit \" kind= \"TripCurrent \" format= \"FLOAT32 \"
defaultValue= \"1.0 \" scaleUnits= \"Amps \"/>" \
406: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \">" \
407: "<Drange name= \"PowerStateEnum \">" \
408: "<Option name= \"Off \" value= \"0 \"/>" \
409: "<Option name= \"On \" value= \"1 \"/>" \
410: "<Option name= \"Tripped \" value= \"2 \"/>" \
411: "</Drange>" \
412: "</Variable>" \
413: "<Notification>" \
414: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \

```


415: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \\
 416: "<VariableRef name= \"Time \" />\" \\
 417: "<VariableRef name= \"SubS \" />\" \\
 418: "<VariableRef name= \"PortPowerState \" />\" \\
 419: "<VariableRef name= \"PortVoltage \"/>\" \\
 420: "<VariableRef name= \"PortCurrent \"/>\" \\
 421: "</DataMsg>\" \\
 422: "</Notification>\" \\
 423: "<Notification>\" \\
 424: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \">\" \\
 425: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \\
 426: "<VariableRef name= \"Time \" />\" \\
 427: "<VariableRef name= \"SubS \" />\" \\
 428: "<VariableRef name= \"PortPowerState \"/>\" \\
 429: "</DataMsg>\" \\
 430: "</Notification>\" \\
 431: "<Command>\" \\
 432: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \">\" \\
 433: "<VariableRef name= \"SoftCurrentLimit \"/>\" \\
 434: "</CommandMsg>\" \\
 435: "</Command>\" \\
 436: "<Command>\" \\
 437: "<CommandMsg id= \"4 \" name= \"PortPowerOn \" />\" \\
 438: "</Command>\" \\
 439: "<Command>\" \\
 440: "<CommandMsg id= \"5 \" name= \"PortPowerOff \" />\" \\
 441: "</Command>\" \\
 442: "</Interface>\" \\
 443: \"\" \\
 444: \"\" \\
 445: "<Interface id= \"11 \" name= \"ASIMPortInterface \" description= \"Interface for a single SPA-U port on the hub. \">\" \\
 446: "<Qualifier name= \"PortID \" value= \"6 \"/>\" \\
 447: "<Qualifier name= \"BreakerTripCurrent \" value= \"4.5 \" units= \"Amps \"/>\" \\
 448: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \\
 449: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \\
 450: "<Variable name= \"PortVoltage \" kind= \"Voltage \" format= \"FLOAT32 \" scaleUnits= \"Volts \"/>\" \\
 451: "<Variable name= \"PortCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps \"/>\" \

```

452: "<Variable name= \"SoftCurrentLimit \" kind= \"TripCurrent \" format= \"FLOAT32 \"
defaultValue= \"1.0 \" scaleUnits= \"Amps \"/>\" \
453: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \"/>\" \
454: "<Drange name= \"PowerStateEnum \"/>\" \
455: "<Option name= \"Off \" value= \"0 \"/>\" \
456: "<Option name= \"On \" value= \"1 \"/>\" \
457: "<Option name= \"Tripped \" value= \"2 \"/>\" \
458: "</Drange>\" \
459: "</Variable>\" \
460: "<Notification>\" \
461: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">\" \
462: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
463: "<VariableRef name= \"Time \" />\" \
464: "<VariableRef name= \"SubS \" />\" \
465: "<VariableRef name= \"PortPowerState \" />\" \
466: "<VariableRef name= \"PortVoltage \"/>\" \
467: "<VariableRef name= \"PortCurrent \"/>\" \
468: "</DataMsg>\" \
469: "</Notification>\" \
470: "<Notification>\" \
471: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \"/>\" \
472: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
473: "<VariableRef name= \"Time \" />\" \
474: "<VariableRef name= \"SubS \" />\" \
475: "<VariableRef name= \"PortPowerState \"/>\" \
476: "</DataMsg>\" \
477: "</Notification>\" \
478: "<Command>\" \
479: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \"/>\" \
480: "<VariableRef name= \"SoftCurrentLimit \"/>\" \
481: "</CommandMsg>\" \
482: "</Command>\" \
483: "<Command>\" \
484: "<CommandMsg id= \"4 \" name= \"PortPowerOn \" />\" \
485: "</Command>\" \
486: "<Command>\" \
487: "<CommandMsg id= \"5 \" name= \"PortPowerOff \" />\" \
488: "</Command>\" \
489: "</Interface>\" \
490: "" \
491: "" \

```

```

492: "<Interface id= \"12 \" name= \"ASIMPortInterface \" description= \"Interface for a single SPA-U
port on the hub. \"/>" \
493: "<Qualifier name= \"PortID \" value= \"7 \"/>" \
494: "<Qualifier name= \"BreakerTripCurrent \" value= \"4.5 \" units= \"Amps \"/>" \
495: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
496: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
497: "<Variable name= \"PortVoltage \" kind= \"Voltage \" format= \"FLOAT32 \" scaleUnits= \"Volts
\"/>" \
498: "<Variable name= \"PortCurrent \" kind= \"Current \" format= \"FLOAT32 \" scaleUnits= \"Amps
\"/>" \
499: "<Variable name= \"SoftCurrentLimit \" kind= \"TripCurrent \" format= \"FLOAT32 \"
defaultValue= \"1.0 \" scaleUnits= \"Amps \"/>" \
500: "<Variable name= \"PortPowerState \" kind= \"boolean \" format= \"UINT08 \"/>" \
501: "<Drange name= \"PowerStateEnum \"/>" \
502: "<Option name= \"Off \" value= \"0 \"/>" \
503: "<Option name= \"On \" value= \"1 \"/>" \
504: "<Option name= \"Tripped \" value= \"2 \"/>" \
505: "</Drange>" \
506: "</Variable>" \
507: "<Notification>" \
508: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
509: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
510: "<VariableRef name= \"Time \"/>" \
511: "<VariableRef name= \"SubS \"/>" \
512: "<VariableRef name= \"PortPowerState \"/>" \
513: "<VariableRef name= \"PortVoltage \"/>" \
514: "<VariableRef name= \"PortCurrent \"/>" \
515: "</DataMsg>" \
516: "</Notification>" \
517: "<Notification>" \
518: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \"/>" \
519: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
520: "<VariableRef name= \"Time \"/>" \
521: "<VariableRef name= \"SubS \"/>" \
522: "<VariableRef name= \"PortPowerState \"/>" \
523: "</DataMsg>" \
524: "</Notification>" \
525: "<Command>" \
526: "<CommandMsg id= \"3 \" name= \"ConfigureSoftTrip \"/>" \
527: "<VariableRef name= \"SoftCurrentLimit \"/>" \
528: "</CommandMsg>" \

```

```

529: "</Command>" \
530: "<Command>" \
531: "<CommandMsg id= \"4\" name= \"PortPowerOn\" />" \
532: "</Command>" \
533: "<Command>" \
534: "<CommandMsg id= \"5\" name= \"PortPowerOff\" />" \
535: "</Command>" \
536: "</Interface>" \
537: "" \
538: "" \
539: "<Interface name= \"CmpSafety\" id= \"13\">" \
540: "<Qualifier name= \"TemperatureLoKeepout\" value= \"-20.0\" units= \"degC\" />" \
541: "<Qualifier name= \"TemperatureLoWarning\" value= \"-10.0\" units= \"degC\" />" \
542: "<Qualifier name= \"TemperatureHiWarning\" value= \"50.0\" units= \"degC\" />" \
543: "<Qualifier name= \"TemperatureHiKeepout\" value= \"60.0\" units= \"degC\" />" \
544: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />" \
545: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\" \
scaleFactor= \".0001\" scaleUnits= \"Seconds\" />" \
546: "<Variable name= \"DeviceTemperature\" kind= \"temperature\" format= \"FLOAT32\" units= \
\"degC\" />" \
547: "<Request>" \
548: "<CommandMsg name= \"GetDeviceTemperature\" id= \"1\" />" \
549: "<DataReplyMsg name= \"DeviceTempReply\" id= \"2\">" \
550: "<VariableRef name= \"Time\" />" \
551: "<VariableRef name= \"SubS\" />" \
552: "<VariableRef name= \"DeviceTemperature\" />" \
553: "</DataReplyMsg>" \
554: "</Request>" \
555: "<Notification>" \
556: "<DataMsg name= \"DeviceTemp\" id= \"3\" msgArrival= \"PERIODIC\" msgRate= \"1\">" \
557: "<Qualifier name= \"telemetryLevel\" value= \"1\" />" \
558: "<VariableRef name= \"Time\" />" \
559: "<VariableRef name= \"SubS\" />" \
560: "<VariableRef name= \"DeviceTemperature\" />" \
561: "</DataMsg>" \
562: "</Notification>" \
563: "</Interface>" \
564: "" \
565: "</xTEDS>" \
566: ""
567:

```

568: #endif

File: sdm/app/test/DMTests/xTEDSRegTests/ADCSController.h

```
1: #ifndef _ADCSCONTROLLER_XTEDS_H
2: #define _ADCSCONTROLLER_XTEDS_H
3:
4: #define _STRING_ADCSCONTROLLER_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"ADCSControl_App_xTeds\" version= \"2.1\">\" \
8:   \"<Application name= \"ADCSController \" version= \"1.0 \" kind= \"SHAP \" description=
\"Implements the top-level interface to the ADCS \" />\" \
9:   \"<Interface name= \"ADCSInterface \" id= \"1\">\" \
10:  \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
11:  \"<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \" />\" \
12:  \"<Variable name= \"Rate \" kind= \"attitudeRate \" units= \"rad_s \" format= \"FLOAT64 \" length=
\"3 \" description= \"Desired Angular rates about each primary axis of the vehicle \">\" \
13:  \"<Qualifier name= \"representation \" value= \"vector \" />\" \
14:  \"<Qualifier name= \"frameMeasured \" value= \"SVF \" />\" \
15:  \"<Qualifier name= \"frameResolved \" value= \"SVF \" />\" \
16:  \"</Variable>\" \
17:  \"<Variable name= \"TargetObject \" kind= \"target \" format= \"UINT08 \" description= \"Target at
which the system should point \">\" \
18:  \"<Drange name= \"TargetsEnum \">\" \
19:  \"<Option name= \"Nadir \" value= \"0\" />\" \
20:  \"<Option name= \"Zenith \" value= \"1\" />\" \
21:  \"<Option name= \"Sun \" value= \"2\" />\" \
22:  \"<Option name= \"Other \" value= \"3\" />\" \
23:  \"</Drange>\" \
24:  \"</Variable>\" \
25:  \"<Variable name= \"TargetPosition \" kind= \"position \" length= \"3 \" units= \"km \" format=
\"FLOAT64\" description= \"ECI position vector of a target at which we wish to point \">\" \
26:  \"<Qualifier name= \"representation \" value= \"vector \" />\" \
27:  \"<Qualifier name= \"frameMeasured \" value= \"ECIMOD \" />\" \
28:  \"<Qualifier name= \"frameResolved \" value= \"ECIMOD \" />\" \
29:  \"</Variable>\" \
30:  \"<Variable name= \"TargetVelocity \" kind= \"velocity \" length= \"3 \" units= \"m_s \" format=
\"FLOAT64\" description= \"ECI velocity vector of a target at which we wish to point \">\" \
31:  \"<Qualifier name= \"representation \" value= \"vector \" />\" \
32:  \"<Qualifier name= \"frameMeasured \" value= \"ECIMOD \" />\" \
33:  \"<Qualifier name= \"frameResolved \" value= \"ECIMOD \" />\" \
```

```

34: "</Variable>" \
35: "<Variable name= \"TargetAcceleration \" kind= \"acceleration \" length= \"3 \" units= \"m_s_s \"
format= \"FLOAT64 \" description= \"ECI acceleration vector of a target at which we wish to point \">>" \
36: "<Qualifier name= \"representation \" value= \"vector \"/>" \
37: "<Qualifier name= \"frameMeasured \" value= \"ECIMOD \"/>" \
38: "<Qualifier name= \"frameResolved \" value= \"ECIMOD \"/>" \
39: "</Variable>" \
40: "<Variable name= \"BoresightObject \" kind= \"boresight \" format= \"UINT08 \" description=
\"Boresight we wish to point \" >" \
41: "<Drange name= \"BoresightsEnum \">>" \
42: "<Option name= \"SolarPanel \" value= \"0 \"/>" \
43: "<Option name= \"PrimaryPayload \" value= \"1 \"/>" \
44: "<Option name= \"Other \" value= \"2 \"/>" \
45: "</Drange>" \
46: "</Variable>" \
47: "<Variable name= \"BoresightLOS \" kind= \"LOS \" length= \"3 \" format= \"FLOAT64 \"
description= \"LOS of the Boresight we wish to point \">>" \
48: "<Qualifier name= \"representation \" value= \"vector \"/>" \
49: "<Qualifier name= \"frameMeasured \" value= \"SVF \"/>" \
50: "<Qualifier name= \"frameResolved \" value= \"SVF \"/>" \
51: "</Variable>" \
52: "<Variable name= \"Offset \" kind= \"rollPitchYaw \" length= \"3 \" units= \"rad \" format=
\"FLOAT32 \" description= \"pushbroom offsets from LVLH [rad] \"/>" \
53: "<!--Commands for the ADCS System-->" \
54: "<Command>" \
55: "<CommandMsg id= \"1 \" name= \"SetADCSModeStandby \" description= \"Set the ADCS to
standby mode \" />" \
56: "</Command>" \
57: "<Command>" \
58: "<CommandMsg id= \"2 \" name= \"SetADCSModeRate \" description= \"Set the ADCS to rate-only
mode \">>" \
59: "<VariableRef name= \"Rate \"/>" \
60: "</CommandMsg>" \
61: "</Command>" \
62: "<Command>" \
63: "<CommandMsg id= \"3 \" name= \"SetADCSModeTracking \" description= \"Set the ADCS to
tracking mode \">>" \
64: "<VariableRef name= \"BoresightObject \"/>" \
65: "<VariableRef name= \"TargetObject \"/>" \
66: "<VariableRef name= \"Offset \"/>" \
67: "<!-- Boresight LOS only used if BoresightObject is \"other \", send 0's otherwise-->" \
68: "<VariableRef name= \"BoresightLOS \"/>" \

```

```

69: "<!--Target Specifications only used if TargetObject is \"other \", send 0's otherwise-->" \
70: "<VariableRef name= \"TargetPosition \"/>" \
71: "<VariableRef name= \"TargetVelocity \"/>" \
72: "<VariableRef name= \"TargetAcceleration \"/>" \
73: "</CommandMsg>" \
74: "</Command>" \
75: "<Command>" \
76: "<CommandMsg id= \"4 \" name= \"SetADCSModeMomentumDump \" description= \"Set the
ADCS to momentum dumping mode \" />" \
77: "</Command>" \
78: "</Interface>" \
79: "</xTEDS>" \
80: ""
81:
82: #endif

```


File: sdm/app/test/DMTests/xTEDSRegTests/ActivityManager.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS    ../Schema/xTEDS02.xsd"
name="ActivityManagerXTEDS" description="ActivityManager xTEDS" version="3.1">
4:
5:     <Application name="ActivityManager" kind="AutonomyFlightSoftware" description="Autonomous
Tasking Executive (ATE), ActivityManager"/>
6:
7:     <Interface name="ActivityManagerInterface" id="1" description="Basic interface for scheduling
activities and updating their status">
8:
9:         <Variable name="ActivityId" kind="ID" format="UINT32"/>
10:        <Variable name="ActivityName" kind="String" format="INT08" length="33"/><!-- 33-byte, null
terminated string -->
11:        <Variable name="ActivityPriority" kind="tbd" format="UINT08" rangeMin="0"
rangeMax="255"/><!-- 0 is the lowest priority and 255 is the highest -->
12:        <Variable name="ActivityBeginTime" kind="Time" format="FLOAT64" units="s"/>
13:        <Variable name="ActivityDuration" kind="Time" format="FLOAT64" units="s"/>
14:        <Variable name="ActivityNotBeforeTime" kind="Time" format="FLOAT64" units="s"/><!--
Earliest begin time. 0.0 signifies now -->
15:        <Variable name="ActivityNotAfterTime" kind="Time" format="FLOAT64" units="s"/><!-- Latest
begin time. 0.0 signifies no limit -->
16:        <Variable name="AllowConcurrentActivity" kind="boolean" format="UINT08">
17:            <Drange name="AllowConcurrentActivityEnum">
18:                <Option value="0" name="NO"/>
19:                <Option value="1" name="YES"/>
20:            </Drange>
21:        </Variable>
22:        <Variable name="ActivityStatus" kind="Status" format="INT16">
23:            <Drange name="ActivityStatusEnum">
24:                <Option value="0" name="SCHEDULE_FAILURE"/><!-- The activity could not be scheduled
as requested -->
25:                <Option value="1" name="WAITING"/><!-- The activity has been inserted into the schedule -->
26:                <Option value="2" name="ENABLED"/><!-- The activity has been sent a command to execute -
->
27:                <Option value="3" name="TERMINATED"/><!-- The activity has been sent a command to
abort -->
28:                <Option value="4" name="EXECUTING"/><!-- The activity has indicated that it is currently
executing -->
29:                <Option value="5" name="DONE_FAILURE"/><!-- The activity has indicated that it completed
abnormally -->
```

```

30:     <Option value="6" name="DONE_SUCCESS"/><!-- The activity has indicated that it completed
normally -->
31:     <Option value="7" name="DONE_NOT_EXECUTED"/><!-- The activity has indicated that it
terminated without executing -->
32:     </Drange>
33: </Variable>
34: <Variable name="ExecutionStatus" kind="Status" format="INT16" defaultValue="4">
35:     <Drange name="ExecutionStatusEnum">
36:         <Option value="4" name="EXECUTING"/><!-- The activity is currently executing -->
37:         <Option value="5" name="DONE_FAILURE"/><!-- The activity has completed abnormally -->
38:         <Option value="6" name="DONE_SUCCESS"/><!-- The activity has completed normally -->
39:         <Option value="7" name="DONE_NOT_EXECUTED"/><!-- The activity has terminated
without executing -->
40:     </Drange>
41: </Variable>
42:
43: <!-- Variables for resources required are tbd -->
44:
45: <Request>
46:     <CommandMsg name="ScheduleActivity" id="001" description="Schedule a mission or
housekeeping activity">
47:         <VariableRef name="ActivityId"/>
48:         <VariableRef name="ActivityName"/>
49:         <VariableRef name="ActivityPriority"/>
50:         <VariableRef name="ActivityDuration"/>
51:         <VariableRef name="AllowConcurrentActivity"/>
52:         <VariableRef name="ActivityNotBeforeTime"/>
53:         <VariableRef name="ActivityNotAfterTime"/>
54:
55:         <!-- VariableRefs for resources are tbd -->
56:
57:     </CommandMsg>
58:     <DataReplyMsg name="ScheduleActivityReply" id="002">
59:         <VariableRef name="ActivityId"/>
60:         <VariableRef name="ActivityStatus"/>
61:     </DataReplyMsg>
62: </Request>
63:
64: <Request>
65:     <CommandMsg name="UpdateActivity" id="003" description="Adjust the properties of a
scheduled mission or housekeeping activity">
66:         <VariableRef name="ActivityId"/>

```

```

67:     <VariableRef name="ActivityPriority"/>
68:     <VariableRef name="ActivityDuration"/>
69:     <VariableRef name="AllowConcurrentActivity"/>
70:     <VariableRef name="ActivityNotBeforeTime"/>
71:     <VariableRef name="ActivityNotAfterTime"/>
72:
73:     <!-- VariableRefs for resources are tbd -->
74:
75: </CommandMsg>
76: <DataReplyMsg name="UpdateActivityReply" id="004">
77:     <VariableRef name="ActivityId"/>
78:     <VariableRef name="ActivityStatus"/>
79: </DataReplyMsg>
80: </Request>
81:
82: <Command>
83:     <CommandMsg name="UpdateActivityStatus" id="005" description="Update activity execution
status">
84:         <VariableRef name="ActivityId"/>
85:         <VariableRef name="ExecutionStatus"/>
86:     </CommandMsg>
87: </Command>
88:
89: <Command>
90:     <CommandMsg name="DeleteActivity" id="006" description="Delete a scheduled activity">
91:         <VariableRef name="ActivityId"/>
92:     </CommandMsg>
93: </Command>
94:
95: <Command>
96:     <CommandMsg name="SendActivityStatusMsg" id="007" description="Send the
ActivityStatusMsg DataMsg">
97:         <VariableRef name="ActivityId"/>
98:     </CommandMsg>
99: </Command>
100:
101: <Notification>
102:     <DataMsg name="ActivityStatusMsg" id="008" msgArrival="EVENT">
103:         <Qualifier value="1" name="telemetryLevel"/>
104:         <VariableRef name="ActivityId"/>
105:         <VariableRef name="ActivityName"/>

```

```

106:     <VariableRef name="ActivityPriority"/>
107:     <VariableRef name="ActivityBeginTime"/>
108:     <VariableRef name="ActivityDuration"/>
109:     <VariableRef name="AllowConcurrentActivity"/>
110:     <VariableRef name="ActivityNotBeforeTime"/>
111:     <VariableRef name="ActivityNotAfterTime"/>
112:     <VariableRef name="ActivityStatus"/>
113: </DataMsg>
114: </Notification>
115:
116: </Interface>
117:
118: <Interface name="ActivityManagerScheduleInterface" id="2" description="Additional messages
for schedule maintenance" >
119:
120:     <Command>
121:         <CommandMsg name="DeleteAllActivities" id="001" description="Delete all scheduled
activities"/>
122:     </Command>
123:
124:     <!-- Command -->
125:     <!-- CommandMsg name="PrepareScheduleFile" id="002" description="Prepare an activities
schedule for downlink" -->
126:     <!-- /Command -->
127:
128: </Interface>
129:
130: <Interface id="3" name="ActivityManagerStatusInterface">
131:
132:     <Variable name="ActivityManagerStatus" kind="Status" format="INT16">
133:         <Drange name="ActivityManagerStatusEnum">
134:             <Option value="0" name="NOT_INITIALIZED"/><!-- The ActivityManager has not been
successfully initialized -->
135:             <Option value="1" name="INITIALIZING"/><!-- The ActivityManager is in the process of
initializing -->
136:             <Option value="2" name="RUNNING"/><!-- The ActivityManager is initialized and is running
-->
137:             <Option value="3" name="TERMINATING"/><!-- The ActivityManager is shutting down -->
138:         </Drange>
139:     </Variable>
140:
141:     <Variable name="ActivitiesCurrentlyScheduled" kind="tbd" format="UINT16"/>

```

```

142: <Variable name="ActivitiesExecuted" kind="tbd" format="UINT16"/>
143: <Variable name="ActivitiesExecutedSuccess" kind="tbd" format="UINT16"/>
144: <Variable name="ActivitiesExecutedFailed" kind="tbd" format="UINT16"/>
145: <Variable name="ActivitiesDeleted" kind="tbd" format="UINT16"/>
146:
147: <Variable name="ScheduleActivityReceived" kind="tbd" format="UINT16"/>
148: <Variable name="ScheduleActivityAccepted" kind="tbd" format="UINT16"/>
149: <Variable name="ScheduleActivitySuccess" kind="tbd" format="UINT16"/>
150: <Variable name="ScheduleActivityFailure" kind="tbd" format="UINT16"/>
151:
152: <Variable name="UpdateActivityReceived" kind="tbd" format="UINT16"/>
153: <Variable name="UpdateActivityAccepted" kind="tbd" format="UINT16"/>
154: <Variable name="UpdateActivitySuccess" kind="tbd" format="UINT16"/>
155: <Variable name="UpdateActivityFailure" kind="tbd" format="UINT16"/>
156:
157: <Variable name="UpdateActivityStatusReceived" kind="tbd" format="UINT16"/>
158: <Variable name="UpdateActivityStatusAccepted" kind="tbd" format="UINT16"/>
159: <Variable name="UpdateActivityStatusSuccess" kind="tbd" format="UINT16"/>
160: <Variable name="UpdateActivityStatusFailure" kind="tbd" format="UINT16"/>
161:
162: <Variable name="DeleteActivityReceived" kind="tbd" format="UINT16"/>
163: <Variable name="DeleteActivityAccepted" kind="tbd" format="UINT16"/>
164: <Variable name="DeleteActivitySuccess" kind="tbd" format="UINT16"/>
165: <Variable name="DeleteActivityFailure" kind="tbd" format="UINT16"/>
166:
167: <Variable name="DeleteAllActivitiesReceived" kind="tbd" format="UINT16"/>
168: <Variable name="DeleteAllActivitiesAccepted" kind="tbd" format="UINT16"/>
169: <Variable name="DeleteAllActivitiesSuccess" kind="tbd" format="UINT16"/>
170: <Variable name="DeleteAllActivitiesFailure" kind="tbd" format="UINT16"/>
171:
172: <!-- Variable name="PrepSchedFileReceived" kind="tbd" format="UINT16"/ -->
173: <!-- Variable name="PrepSchedFileAccepted" kind="tbd" format="UINT16"/ -->
174: <!-- Variable name="PrepSchedFileSuccess" kind="tbd" format="UINT16"/ -->
175: <!-- Variable name="PrepSchedFileFailure" kind="tbd" format="UINT16"/ -->
176:
177: <Command>
178:     <CommandMsg name="SendActivityManagerStatusMsg" id="001" description="Send the
ActivityManagerStatusMsg DataMsg"/>
179: </Command>
180:
181: <Notification>

```

```

182: <DataMsg name="ActivityManagerStatusMsg" id="002" msgArrival="EVENT">
183:   <Qualifier value="1" name="telemetryLevel"/>
184:   <VariableRef name="ActivityManagerStatus"/>
185:   <VariableRef name="ActivitiesCurrentlyScheduled"/>
186:   <VariableRef name="ActivitiesExecuted"/>
187:   <VariableRef name="ActivitiesExecutedSuccess"/>
188:   <VariableRef name="ActivitiesExecutedFailed"/>
189:   <VariableRef name="ActivitiesDeleted"/>
190:   <VariableRef name="ScheduleActivityReceived"/>
191:   <VariableRef name="ScheduleActivityAccepted"/>
192:   <VariableRef name="ScheduleActivitySuccess"/>
193:   <VariableRef name="ScheduleActivityFailure"/>
194:   <VariableRef name="UpdateActivityReceived"/>
195:   <VariableRef name="UpdateActivitySuccess"/>
196:   <VariableRef name="UpdateActivityFailure"/>
197:   <VariableRef name="UpdateActivityStatusReceived"/>
198:   <VariableRef name="UpdateActivityStatusAccepted"/>
199:   <VariableRef name="UpdateActivityStatusSuccess"/>
200:   <VariableRef name="UpdateActivityStatusFailure"/>
201:   <VariableRef name="DeleteActivityReceived"/>
202:   <VariableRef name="DeleteActivityAccepted"/>
203:   <VariableRef name="DeleteActivitySuccess"/>
204:   <VariableRef name="DeleteActivityFailure"/>
205:   <VariableRef name="DeleteAllActivitiesReceived"/>
206:   <VariableRef name="DeleteAllActivitiesAccepted"/>
207:   <VariableRef name="DeleteAllActivitiesSuccess"/>
208:   <VariableRef name="DeleteAllActivitiesFailure"/>
209:   <!-- VariableRef name="PrepSchedFileReceived"/ -->
210:   <!-- VariableRef name="PrepSchedFileAccepted"/ -->
211:   <!-- VariableRef name="PrepSchedFileSuccess"/ -->
212:   <!-- VariableRef name="PrepSchedFileFailure"/ -->
213: </DataMsg>
214: </Notification>
215:
216: </Interface>
217:
218: <Interface name="ICSDebugInterface" id="4">
219:
220:   <!--
221:   Note: DebugLevel is a bit field with the following assigned values:
222:   DEBUG_ENTRY_AND_EXIT = 0x01,

```

```

223:     DEBUG_ENTRY_PARAMETERS = 0x02,
224:     DEBUG_EXIT_PARAMETERS  = 0x04,
225:     DEBUG_LEVEL_LOW       = 0x08,
226:     DEBUG_LEVEL_MEDIUM    = 0x10,
227:     DEBUG_LEVEL_HIGH      = 0x20.
228:     The values are OR'd to determine the debug information to be logged.
229: -->
230:
231:     <Variable name="DebugLevel" kind="tbd" format="UINT16"/>
232:     <Variable name="CurrentDebugLevel" kind="tbd" format="UINT16"/>
233:
234:     <Variable name="SetDebugLevelReceived" kind="tbd" format="UINT16"/>
235:     <Variable name="SetDebugLevelAccepted" kind="tbd" format="UINT16"/>
236:     <Variable name="SetDebugLevelSuccess" kind="tbd" format="UINT16"/>
237:     <Variable name="SetDebugLevelFailure" kind="tbd" format="UINT16"/>
238:
239:     <Command>
240:         <CommandMsg name="SetDebugLevel" id="001" description="Set the debug log verbosity
level">
241:             <VariableRef name="DebugLevel"/>
242:         </CommandMsg>
243:     </Command>
244:
245:     <Command>
246:         <CommandMsg name="SendDebugStatus" id="002"/>
247:     </Command>
248:
249:     <Notification>
250:         <DataMsg name="DebugStatus" id="003" msgArrival="EVENT">
251:             <Qualifier value="1" name="telemetryLevel"/>
252:             <VariableRef name="CurrentDebugLevel"/>
253:             <VariableRef name="SetDebugLevelReceived"/>
254:             <VariableRef name="SetDebugLevelAccepted"/>
255:             <VariableRef name="SetDebugLevelSuccess"/>
256:             <VariableRef name="SetDebugLevelFailure"/>
257:         </DataMsg>
258:     </Notification>
259:
260: </Interface>
261:
262: <Interface name="ICSTaskControlInterface" id="5">

```

263:
264: <Command>
265: <CommandMsg name="DestroyTask" id="001"/>
266: </Command>
267:
268: <!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->
269: <!-- Other possible requests include GetPriority, and GetHeartBeatCount -->
270:
271: </Interface>
272:
273: </xTEDS>

File: sdm/app/test/DMTests/xTEDSRegTests/PowerController.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="Battery_XTEDS"
4:   version="2.0">
5:   <Application name="PowerController" kind="PowerControl" description="Routine Control of
Spacecraft Power" >
6:       <Qualifier name="Author" value="Design_Net"/>
7:       <Qualifier name="Version" value="1.1"/>
8:       <Qualifier name="RevisionDate" value="03-27-2007"/>
9:   </Application>
10:
11:   <Interface name="PowerStatusInterface" id="1">
12:       <Variable name="Time" kind="Time" format="UINT32" units="Seconds" />
13:       <Variable name="SubS" kind="SubSeconds" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
14:       <Variable name="BusVoltage" kind="BusVoltage" format="FLOAT32" units="Volts"
description="Spacecraft Bus Voltage" />
15:       <Variable name="EnergyStoreCapacity" kind="Capacity" format="FLOAT32" units="Watt-Hrs"
description="Energy Storage Capacity">
16:           <Qualifier name="Scope" value="System"/>
17:           <Qualifier name="Category" value="Max"/>
18:       </Variable>
19:       <Variable name="EnergyStorePowerCapacity" kind="Capacity" format="FLOAT32"
units="Watts" description="Energy Storage Power Capacity">
20:           <Qualifier name="Scope" value="System"/>
21:           <Qualifier name="Category" value="Max"/>
22:       </Variable>
23:       <Variable name="CollectionPowerCapacity" kind="Capacity" format="FLOAT32"
units="Watts" description="Energy Collection Power Capacity">
24:           <Qualifier name="Scope" value="System"/>
25:           <Qualifier name="Category" value="Max"/>
26:       </Variable>
27:       <Variable name="DissipationPowerCapacity" kind="Capacity" format="FLOAT32"
units="Watts" description="Energy Dissipation Power Capacity">
28:           <Qualifier name="Scope" value="System"/>
29:           <Qualifier name="Category" value="Max"/>
30:       </Variable>
31:       <Variable name="StoredEnergy" kind="Quantity" format="FLOAT32" units="Watt-Hrs"
description="Energy Store Stored Energy">
```

```

32:     <Qualifier name="Scope" value="System"/>
33:     <Qualifier name="Category" value="Instantaneous"/>
34: </Variable>
35: <Variable name="EnergyStoreStateOfCharge" kind="Percentage" format="FLOAT32"
units="Percent" description="Energy Store State Of Charge">
36:     <Qualifier name="Scope" value="System"/>
37:     <Qualifier name="Category" value="Instantaneous"/>
38: </Variable>
39: <Variable name="EnergyStorePower" kind="Power" format="FLOAT32" units="Watts"
description="Energy Store Power">
40:     <Qualifier name="Scope" value="System"/>
41:     <Qualifier name="Category" value="Instantaneous"/>
42: </Variable>
43: <Variable name="CollectionPower" kind="Power" format="FLOAT32" units="Watts"
description="Energy Collection Power">
44:     <Qualifier name="Scope" value="System"/>
45:     <Qualifier name="Category" value="Instantaneous"/>
46: </Variable>
47: <Variable name="DissipationPower" kind="Power" format="FLOAT32" units="Watts"
description="Energy Dissipation Power">
48:     <Qualifier name="Scope" value="System"/>
49:     <Qualifier name="Category" value="Instantaneous"/>
50: </Variable>
51: <Variable name="DevicePower" kind="Power" format="FLOAT32" units="Watts"
description="Device Power">
52:     <Qualifier name="Scope" value="System"/>
53:     <Qualifier name="Category" value="Instantaneous"/>
54: </Variable>
55: <Variable name="TimeToYellowLimit" kind="Duration" format="UINT32" units="Seconds"
description="Time until Yellow Limit Condition is Reached" >
56:     <Qualifier name="Scope" value="PowerController"/>
57:     <Qualifier name="Target" value="PowerAdvisoryCondition"/>
58: </Variable>
59: <Variable name="TimeToRedLimit" kind="Duration" format="UINT32" units="Seconds"
description="Time until Red Limit Condition is Reached" >
60:     <Qualifier name="Scope" value="PowerManagement"/>
61:     <Qualifier name="Target" value="PowerWarningCondition"/>
62: </Variable>
63: <Variable name="SensorId" kind="Identification" format="UINT32" description="Unique
identifier of spacecraft component"/>
64: <Variable name="HubNumber" kind="Identification" format="UINT32" description="Hub
number"/>

```

```

65:    <Variable name="PortNumber" kind="Identification" format="UINT32" description="Port
number"/>
66:
67:    <!-- Notifications -->
68:    <Notification>
69:        <DataMsg name="PowerState" description="System-Level Power State" id="1"
msgArrival="PERIODIC">
70:            <Qualifier name="telemetryLevel" value="1"/>
71:            <VariableRef name="Time" />
72:            <VariableRef name="SubS" />
73:            <VariableRef name="BusVoltage" />
74:            <VariableRef name="EnergyStoreCapacity" />
75:            <VariableRef name="EnergyStorePowerCapacity" />
76:            <VariableRef name="CollectionPowerCapacity" />
77:            <VariableRef name="DissipationPowerCapacity" />
78:            <VariableRef name="StoredEnergy" />
79:            <VariableRef name="EnergyStoreStateOfCharge" />
80:            <VariableRef name="EnergyStorePower" />
81:            <VariableRef name="CollectionPower" />
82:            <VariableRef name="DissipationPower" />
83:            <VariableRef name="DevicePower" />
84:        </DataMsg>
85:    </Notification>
86:    <Notification>
87:        <DataMsg name="YellowLimitState" description="Time remaining until yellow limit event"
id="2" msgArrival="PERIODIC">
88:            <Qualifier name="telemetryLevel" value="1"/>
89:            <VariableRef name="Time" />
90:            <VariableRef name="SubS" />
91:            <VariableRef name="TimeToYellowLimit"/>
92:        </DataMsg>
93:    </Notification>
94:    <Notification>
95:        <DataMsg name="RedLimitState" description="Time remaining until red limit event" id="3"
msgArrival="PERIODIC">
96:            <Qualifier name="telemetryLevel" value="1"/>
97:            <VariableRef name="Time" />
98:            <VariableRef name="SubS" />
99:            <VariableRef name="TimeToRedLimit"/>
100:        </DataMsg>
101:    </Notification>
102:    <Notification>

```

```

103:         <DataMsg name="YellowLimitEvent" description="Yellow limit condition notification"
id="4" msgArrival="EVENT">
104:             <VariableRef name="Time" />
105:             <VariableRef name="SubS" />
106:             <VariableRef name="EnergyStoreStateOfCharge" />
107:         </DataMsg>
108:     </Notification>
109: </Notification>
110:     <DataMsg name="RedLimitEvent" description="Red limit condition notification" id="5"
msgArrival="EVENT">
111:         <VariableRef name="Time" />
112:         <VariableRef name="SubS" />
113:         <VariableRef name="EnergyStoreStateOfCharge" />
114:     </DataMsg>
115: </Notification>
116: </Notification>
117:     <DataMsg name="PortSoftTripEvent" description="Power hub port soft current trip
notification" id="6" msgArrival="EVENT">
118:         <VariableRef name="Time" />
119:         <VariableRef name="SubS" />
120:         <VariableRef name="SensorId" />
121:         <VariableRef name="HubNumber" />
122:         <VariableRef name="PortNumber" />
123:     </DataMsg>
124: </Notification>
125: </Notification>
126:     <DataMsg name="PortHardTripEvent" description="Power hub port hard current trip
notification" id="7" msgArrival="EVENT">
127:         <VariableRef name="Time" />
128:         <VariableRef name="SubS" />
129:         <VariableRef name="SensorId" />
130:         <VariableRef name="HubNumber" />
131:         <VariableRef name="PortNumber" />
132:     </DataMsg>
133: </Notification>
134:
135: <!-- Commands -->
136: <Command>
137:     <CommandMsg name="ResetSoftTrippedPort" id="8">
138:         <VariableRef name="HubNumber" />
139:         <VariableRef name="PortNumber" />
140:     </CommandMsg>

```

```

141:     </Command>
142:     <Command>
143:         <CommandMsg name="ResetHardTrippedPort" id="9">
144:             <VariableRef name="HubNumber" />
145:             <VariableRef name="PortNumber" />
146:         </CommandMsg>
147:     </Command>
148: </Interface>
149:
150: <Interface name="PowerPlanningInterface" id="2">
151:     <Variable name="PlanId" kind="Identification" format="UINT32" description="Unique
152: identifier of planning request"/>
153:     <Variable name="Timeout" kind="Duration" format="UINT32" description="Time until
154: plan expires if not acknowledged"/>
155:     <Variable name="NewTimeToYellowLimit" kind="Duration" format="UINT32"
156: units="Seconds" description="Time until Yellow Limit Condition is Reached" />
157:     <Variable name="NewTimeToRedLimit" kind="Duration" format="UINT32"
158: units="Seconds" description="Time until Red Limit Condition is Reached" />
159:
160:     <!-- Requests -->
161:     <Request>
162:         <CommandMsg name="RequestPower" id="1">
163:
164:         </CommandMsg>
165:         <DataReplyMsg name="RequestPowerReply" id="2">
166:             <VariableRef name="PlanId"/>
167:             <VariableRef name="NewTimeToYellowLimit"/>
168:             <VariableRef name="NewTimeToRedLimit"/>
169:         </DataReplyMsg>
170:     </Request>
171:
172:     <!-- Commands -->
173:     <Command>
174:         <CommandMsg name="AcknowledgePlan" id="3">
175:             <VariableRef name="PlanId"/>
176:         </CommandMsg>
177:     </Command>
178:     <Command>
179:         <CommandMsg name="CancelPlan" id="4">
180:             <VariableRef name="PlanId"/>
181:         </CommandMsg>
182:     </Command>

```

179: </Interface>
180:
181: </xTEDS>

File: sdm/app/test/DMTests/xTEDSRegTests/AdcoleDigitalSS.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd"
name="Adcole_Digital_Sun_Sensor" version="1.0"><!-- plc changed xTEDS schema location -->
4:         <Device name="Adcole_NRL_Digital_Sun_Sensor" kind="Sun_Sensor" modelId="DSS"
description="Adcole Digital 2-Axis Sun Sensor"/>
5:
6: <Interface name="Sun_Sensor_Interface" id="1"><!-- plc added Interface element -->
7:
8: <!-- Timestamp -->
9:         <Variable kind="Time" name="Time" format="UINT32" units="Seconds"><!-- plc changed
FORMAT value -->
10:         </Variable>
11:         <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds"><!-- plc changed FORMAT value -->
12:         </Variable>
13:
14:
15: <!-- Data Definitions -->
16:         <Variable name="Data_Rate" id="3" kind="Msg_Rate" rangeMax="10000" rangeMin="0"
units="Hz" format="INT16"/><!-- plc changed FORMAT value -->
17:         <Variable name="ID" id="4" kind="ID" format="UINT08"/><!-- plc changed FORMAT value -->
18:
19:         <Variable name="Sensor_Number" id="5" kind="TBD" units="Counts" format="UINT16"/><!--
plc changed FORMAT value -->
20:         <Variable name="Sun_Presence" id="6" kind="TBD" units="Counts" format="UINT16"
rangeMin="0" rangeMax="1"/><!-- plc changed FORMAT value -->
21:         <Variable name="Sun_Angle_X" id="7" kind="Angle" units="Degrees" format="FLOAT32"
rangeMin="-128" rangeMax="128" accuracy="0.25" /><!-- plc changed FORMAT value -->
22:         <Variable name="Sun_Angle_Y" id="8" kind="Angle" units="Degrees" format="FLOAT32"
rangeMin="-128" rangeMax="128" accuracy="0.25" /><!-- plc changed FORMAT value -->
23:
24: <!-- Data Messages -->
25:
26: <Notification><!-- plc added Notification element -->
27:         <DataMsg id="2" name="Sun_Angle" msgArrival="PERIODIC" msgRate="100">
28:         <VariableRef name="SubS"/>
29:         <VariableRef name="Time"/>
30:         <VariableRef name="Sensor_Number"/>
31:         <VariableRef name="Sun_Presence"/>
```

```

32:     <VariableRef name="Sun_Angle_X"/>
33:     <VariableRef name="Sun_Angle_Y"/>
34: </DataMsg>
35: </Notification>
36:
37: <!-- Command Messages -->
38: <Command><!-- plc added Command element -->
39:     <CommandMsg id="10" name="Power_Off"/>
40: </Command>
41:
42: <Command><!-- plc added Command element -->
43:     <CommandMsg id="11" name="Power_On"/>
44: </Command>
45:
46: <Command><!-- plc added Command element -->
47:     <CommandMsg id="12" name="Message_Rate">
48:         <VariableRef name="ID"/>
49:         <VariableRef name="Data_Rate" />
50:     </CommandMsg>
51: </Command>
52:
53: </Interface>
54:
55: <Interface name="DevPwr" id="2">
56:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
57:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
58:     <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
59:     <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
60:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
61:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
62:     <Variable name="DevPwrState" kind="Power_State" format="UINT08">
63:         <Drange name="DevPwrStateEnum">
64:             <Option name="DevPwrOFF" value="0" />
65:             <Option name="DevPwrON" value="1" />
66:         </Drange>
67:     </Variable>
68:     <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
69:         <Drange name="DevPwrStateEnumReference" >
70:             <Option name="DevPwrOFF" value="0" />
71:             <Option name="DevPwrON" value="1" />

```



```

72:     </Drange>
73: </Variable>
74: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
75: <Command>
76:     <CommandMsg name="DevPwrSetState" id="1">
77:         <VariableRef name="DevPwrStateSet" />
78:     </CommandMsg>
79: </Command>
80: <Notification>
81:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
82:         <Qualifier name="telemetryLevel" value="1"/>
83:         <VariableRef name="Time" />
84:         <VariableRef name="SubS" />
85:         <VariableRef name="DevPwrState" />
86:         <VariableRef name="DevPwrStateSet" />
87:     </DataMsg>
88: </Notification>
89: <Request>
90:     <CommandMsg name="getPowerInMode" id="4" />
91:     <DataReplyMsg name="powerInMode" id="5">
92:         <VariableRef name="modePowers"/>
93:     </DataReplyMsg>
94: </Request>
95: <Request>
96:     <CommandMsg name="getPowerInMode2" id="4" />
97:     <DataReplyMsg name="powerInMode" id="5">
98:         <VariableRef name="modePowers"/>
99:     </DataReplyMsg>
100:     <FaultMsg name="TestFault" id="12" />
101: </Request>
102: <Notification>
103:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC" >
104:         <VariableRef name="modePowers"/>
105:     </DataMsg>
106:     <FaultMsg name="TestFault" id="12" />
107: </Notification>
108: </Interface>
109: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/RWheelSingle.h

```
1: #ifndef _SINGLE_REACTION_WHEEL_XTEDS_H
2: #define _SINGLE_REACTION_WHEEL_XTEDS_H
3:
4: #define _STRING_SINGLE_REACTION_WHEEL_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   \"<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"Reaction_Wheel_XTEDS \"\" \
8:   \"version= \"2.1 \">\" \
9:   \"<Device name= \"Single_Reaction_Wheel \" kind= \"rWheel \" description= \"An xTeds for a single
reaction wheel using the 2.0 xTeds schema and the CDD \"/>\" \
10:   \"\" \
11:   \"<Interface name= \"RWSingleInterface \" id= \"1 \">\" \
12:   \"<Qualifier name= \"MaxMomentum \" value= \"5.0 \" units= \"Nms \"/>\" \
13:   \"<Qualifier name= \"NominalMomentum \" value= \"2.0 \" units= \"Nms \"/>\" \
14:   \"<Qualifier name= \"TorqueLossMomentum \" value= \"1.0 \" units= \"Nms \"/>\" \
15:   \"<Qualifier name= \"MaxTorqueAtMaxSpeed \" value= \"10.0 \" units= \"Nm \"/>\" \
16:   \"<Qualifier name= \"MaxTorqueAtNominalSpeed \" value= \"5.0 \" units= \"Nm \"/>\" \
17:   \"<Qualifier name= \"TimeConstant \" value= \"0.1 \" units= \"s \"/>\" \
18:   \"<Qualifier name= \"IdlePower \" value= \"1.0 \" units= \"W \"/>\" \
19:   \"<Qualifier name= \"MaxPower \" value= \"10.0 \" units= \"W \"/>\" \
20:   \"<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>\" \
21:   \"<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \"/>\" \
22:   \"<Variable name= \"opMode \" kind= \"mode \" format= \"UINT16 \">\" \
23:   \"<Drange name= \"enumModes \">\" \
24:   \"<Option name= \"idle \" value= \"0 \"/>\" \
25:   \"<Option name= \"operating \" value= \"1 \"/>\" \
26:   \"</Drange>\" \
27:   \"</Variable>\" \
28:   \"<Variable name= \"dataQuality \" kind= \"DataQuality \" format= \"UINT08 \">\" \
29:   \"<Drange name= \"DataQualityEnum \">\" \
30:   \"<Option name= \"dataBad \" value= \"0 \" description= \"data is garbage or NaN. \" />\" \
31:   \"<Option name= \"dataPoor \" value= \"1 \" description= \"data quality is poor. \" />\" \
32:   \"<Option name= \"dataGood \" value= \"2 \" description= \"data is good. \" />\" \
33:   \"</Drange>\" \
34:   \"</Variable>\" \
35:   \"<Variable name= \"commandedTorque \" kind= \"torque \" format= \"FLOAT32 \" units= \"Nm \"
description= \"Magnitude of torque requested of this device \"/>\" \
```

```

36: "<Variable name= \"currentMomentum \" kind= \"angularMomentum \" format= \"FLOAT32 \"
units= \"Nms \" description= \"The current momentum of this wheel. \" />\" \
37: "<Variable name= \"satPercentage \" kind= \"saturationLevel \" format= \"FLOAT32 \" units=
\"percent \" description= \"The current percentage of saturation of this wheel. \" />\" \
38: "<Command>\" \
39: "<!-- note: the name of a command uniquely identifies it within the dictionary -->\" \
40: "<CommandMsg name= \"SingleAxisTorqueCmd \" id= \"1 \">\" \
41: "<VariableRef name= \"commandedTorque \" />\" \
42: "</CommandMsg>\" \
43: "</Command>\" \
44: "<!-- note: the \"kind \" attribute is the key identifier of a data element in the dictionary -->\" \
45: "<Notification>\" \
46: "<DataMsg name= \"wheelSatLevel \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"2 \">\" \
47: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
48: "<VariableRef name= \"Time \" />\" \
49: "<VariableRef name= \"SubS \" />\" \
50: "<VariableRef name= \"dataQuality \" />\" \
51: "<VariableRef name= \"satPercentage \" />\" \
52: "</DataMsg>\" \
53: "</Notification>\" \
54: "<Notification>\" \
55: "<DataMsg name= \"MomentumCurrent \" msgArrival= \"PERIODIC \" msgRate= \"10 \" id= \"3
\">\" \
56: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
57: "<VariableRef name= \"Time \" />\" \
58: "<VariableRef name= \"SubS \" />\" \
59: "<VariableRef name= \"dataQuality \" />\" \
60: "<VariableRef name= \"currentMomentum \" />\" \
61: "</DataMsg>\" \
62: "</Notification>\" \
63: "<Command>\" \
64: "<CommandMsg name= \"setOpMode \" id= \"4 \">\" \
65: "<VariableRef name= \"opMode \" />\" \
66: "</CommandMsg>\" \
67: "</Command>\" \
68: "<Notification>\" \
69: "<DataMsg name= \"opModeChanged \" id= \"5 \" msgArrival= \"EVENT \">\" \
70: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
71: "<VariableRef name= \"opMode \" />\" \
72: "</DataMsg>\" \
73: "</Notification>\" \
74: "</Interface>\" \

```

```

75: "" \
76: "<Interface name= \"DevPwr\" id= \"4\">\" \
77: "<Qualifier name= \"CurrentLoKeepout\" value= \"0.0\" units= \"A\"/>\" \
78: "<Qualifier name= \"CurrentLoWarning\" value= \"0.0\" units= \"A\"/>\" \
79: "<Qualifier name= \"CurrentHiWarning\" value= \"3.5\" units= \"A\"/>\" \
80: "<Qualifier name= \"CurrentHiKeepout\" value= \"3.5\" units= \"A\"/>\" \
81: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />\" \
82: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\" \
scaleFactor= \".0001\" scaleUnits= \"Seconds\" />\" \
83: "<Variable name= \"DevPwrState\" kind= \"Power_State\" format= \"UINT08\">\" \
84: "<Drange name= \"DevPwrStateEnum\">\" \
85: "<Option name= \"DevPwrOFF\" value= \"0\" description= \"All power to device is turned off.\" />\" \
86: "<Option name= \"DevPwrON\" value= \"1\" description= \"Device may draw full power.\" />\" \
87: "</Drange>\" \
88: "</Variable>\" \
89: "<Variable name= \"DevPwrStateSet\" kind= \"Power_State\" format= \"UINT08\" id= \"2\">\" \
90: "<Drange name= \"DevPwrStateEnumReference\" description= \"This should be a reference to \
DevPwrStateEnumeration.\">\" \
91: "<Option name= \"DevPwrOFF\" value= \"0\" description= \"All power to device is turned off.\" />\" \
92: "<Option name= \"DevPwrON\" value= \"1\" description= \"Device may draw full power.\" />\" \
93: "</Drange>\" \
94: "</Variable>\" \
95: "<Variable name= \"modePowers\" kind= \"power\" format= \"FLOAT32\" units= \"W\" length= \
\"2\" />\" \
96: "<Command>\" \
97: "<CommandMsg name= \"DevPwrSetState\" id= \"1\">\" \
98: "<VariableRef name= \"DevPwrStateSet\" />\" \
99: "</CommandMsg>\" \
100: "<FaultMsg name= \"DevPwrStateNotSet\" id= \"2\">\" \
101: "<VariableRef name= \"Time\" />\" \
102: "<VariableRef name= \"SubS\" />\" \
103: "<VariableRef name= \"DevPwrState\" />\" \
104: "<VariableRef name= \"DevPwrStateSet\" />\" \
105: "</FaultMsg>\" \
106: "</Command>\" \
107: "<Notification>\" \
108: "<DataMsg name= \"DevPwrHK\" id= \"3\" msgArrival= \"PERIODIC\">\" \
109: "<Qualifier name= \"telemetryLevel\" value= \"1\"/>\" \
110: "<VariableRef name= \"Time\" />\" \
111: "<VariableRef name= \"SubS\" />\" \

```

```

112: "<VariableRef name= \"DevPwrState \" />" \
113: "<VariableRef name= \"DevPwrStateSet \" />" \
114: "</DataMsg>" \
115: "</Notification>" \
116: "<Request>" \
117: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
118: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">" \
119: "<VariableRef name= \"modePowers \" />" \
120: "</DataReplyMsg>" \
121: "</Request>" \
122: "" \
123: "</Interface>" \
124: "" \
125: "<Interface name= \"CmpSOH \" id= \"5 \">" \
126: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \" />" \
127: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \" />" \
128: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \" />" \
129: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \" />" \
130: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
131: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \
132: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \" />" \
133: "<Request>" \
134: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />" \
135: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">" \
136: "<VariableRef name= \"Time \" />" \
137: "<VariableRef name= \"SubS \" />" \
138: "<VariableRef name= \"DeviceTemperature \" />" \
139: "</DataReplyMsg>" \
140: "</Request>" \
141: "<Notification>" \
142: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">" \
143: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
144: "<VariableRef name= \"Time \" />" \
145: "<VariableRef name= \"SubS \" />" \
146: "<VariableRef name= \"DeviceTemperature \" />" \
147: "</DataMsg>" \
148: "</Notification>" \
149: "</Interface>" \
150: "" \

```

151: "</xTEDS>" \
152: ""
153:
154: #endif

File: sdm/app/test/DMTests/xTEDSRegTests/RoboHub.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="Robo_Hub_xTEDS" version="11.16">
4:
5: <Device name="RoboHub_8_Port" kind="Robust_Hub" modelId="0001" >
6:   <Qualifier name="Manufacturer" value="DataDesignCorp"/>
7:   <Qualifier name="Model" value="Gen1"/>
8:   <Qualifier name="SerialNumber" value="12345"/>
9: </Device>
10:
11: <Interface id="1" name="InterPanelPortInterface" description="Interface for one of the inter-panel
ports">
12:   <Qualifier name="PortID" value="A"/>
13:   <Qualifier name="BreakerTripCurrent" value="30.0" units="Amps"/>
14:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
15:   <Variable   kind="SubSeconds"   name="SubS"   units="Counts"   format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
16:   <Variable name="BreakerCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
17:   <Variable name="PortPowerState" kind="boolean" format="UINT08">
18:     <Drange name="PowerStateEnum">
19:       <Option name="Off" value="0"/>
20:       <Option name="On" value="1"/>
21:       <Option name="Tripped" value="2"/>
22:     </Drange>
23:   </Variable>
24:   <Notification>
25:     <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
26:       <Qualifier name="telemetryLevel" value="1"/>
27:       <VariableRef name="Time" />
28:       <VariableRef name="SubS" />
29:       <VariableRef name="BreakerCurrent"/>
30:     </DataMsg>
31:   </Notification>
32:   <Notification>
33:     <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
34:       <Qualifier name="telemetryLevel" value="1"/>
35:       <VariableRef name="Time" />
36:       <VariableRef name="SubS" />
```

```

37:         <VariableRef name="PortPowerState"/>
38:     </DataMsg>
39: </Notification>
40: <Command>
41:     <CommandMsg id="2" name="PortPowerOn" />
42: </Command>
43: <Command>
44:     <CommandMsg id="3" name="PortPowerOff" />
45: </Command>
46: </Interface>
47:
48: <Interface id="2" name="InterPanelPortInterface" description="Interface for one of the inter-panel
ports">
49:     <Qualifier name="portID" value="B"/>
50:     <Qualifier name="BreakerTripCurrent" value="30.0" units="Amps"/>
51:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
52:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
53:     <Variable name="BreakerCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
54:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
55:         <Drange name="PowerStateEnum">
56:             <Option name="Off" value="0"/>
57:             <Option name="On" value="1"/>
58:             <Option name="Tripped" value="2"/>
59:         </Drange>
60:     </Variable>
61: <Notification>
62:     <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
63:         <Qualifier name="telemetryLevel" value="1"/>
64:         <VariableRef name="Time" />
65:         <VariableRef name="SubS" />
66:         <VariableRef name="BreakerCurrent"/>
67:     </DataMsg>
68: </Notification>
69: <Notification>
70:     <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
71:         <Qualifier name="telemetryLevel" value="1"/>
72:         <VariableRef name="Time" />
73:         <VariableRef name="SubS" />
74:         <VariableRef name="PortPowerState"/>
75:     </DataMsg>

```



```

76:     </Notification>
77:     <Command>
78:         <CommandMsg id="2" name="PortPowerOn" />
79:     </Command>
80:     <Command>
81:         <CommandMsg id="3" name="PortPowerOff" />
82:     </Command>
83: </Interface>
84:
85: <Interface id="3" name="InterPanelPortInterface" description="Interface for one of the inter-panel
ports">
86:     <Qualifier name="PortID" value="C"/>
87:     <Qualifier name="BreakerTripCurrent" value="30.0" units="Amps"/>
88:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
89:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
90:     <Variable name="BreakerCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
91:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
92:         <Drange name="PowerStateEnum">
93:             <Option name="Off" value="0"/>
94:             <Option name="On" value="1"/>
95:             <Option name="Tripped" value="2"/>
96:         </Drange>
97:     </Variable>
98:     <Notification>
99:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
100:             <Qualifier name="telemetryLevel" value="1"/>
101:             <VariableRef name="Time" />
102:             <VariableRef name="SubS" />
103:             <VariableRef name="BreakerCurrent"/>
104:         </DataMsg>
105:     </Notification>
106:     <Notification>
107:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
108:             <Qualifier name="telemetryLevel" value="1"/>
109:             <VariableRef name="Time" />
110:             <VariableRef name="SubS" />
111:             <VariableRef name="PortPowerState"/>
112:         </DataMsg>
113:     </Notification>
114:     <Command>

```

```

115:         <CommandMsg id="2" name="PortPowerOn" />
116:     </Command>
117:     <Command>
118:         <CommandMsg id="3" name="PortPowerOff" />
119:     </Command>
120: </Interface>
121:
122: <Interface id="4" name="InterPanelPortInterface" description="Interface for one of the inter-
panel ports">
123:     <Qualifier name="PortID" value="D"/>
124:     <Qualifier name="BreakerTripCurrent" value="30.0" units="Amps"/>
125:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
126:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
127:     <Variable name="BreakerCurrent" kind="Current" format="FLOAT32"
scaleUnits="Amps"/>
128:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
129:         <Drange name="PowerStateEnum">
130:             <Option name="Off" value="0"/>
131:             <Option name="On" value="1"/>
132:             <Option name="Tripped" value="2"/>
133:         </Drange>
134:     </Variable>
135:     <Notification>
136:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
137:             <Qualifier name="telemetryLevel" value="1"/>
138:             <VariableRef name="Time" />
139:             <VariableRef name="SubS" />
140:             <VariableRef name="BreakerCurrent"/>
141:         </DataMsg>
142:     </Notification>
143:     <Notification>
144:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
145:             <Qualifier name="telemetryLevel" value="1"/>
146:             <VariableRef name="Time" />
147:             <VariableRef name="SubS" />
148:             <VariableRef name="PortPowerState"/>
149:         </DataMsg>
150:     </Notification>
151:     <Command>
152:         <CommandMsg id="2" name="PortPowerOn" />
153:     </Command>

```

```

154:     <Command>
155:         <CommandMsg id="3" name="PortPowerOff" />
156:     </Command>
157: </Interface>
158:
159: <Interface id="5" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
160:     <Qualifier name="PortID" value="0"/>
161:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>
162:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
163:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
164:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
165:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
166:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
167:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
168:         <Drange name="PowerStateEnum">
169:             <Option name="Off" value="0"/>
170:             <Option name="On" value="1"/>
171:             <Option name="Tripped" value="2"/>
172:         </Drange>
173:     </Variable>
174:     <Notification>
175:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
176:             <Qualifier name="telemetryLevel" value="1"/>
177:             <VariableRef name="Time" />
178:             <VariableRef name="SubS" />
179:             <VariableRef name="PortPowerState" />
180:             <VariableRef name="PortVoltage"/>
181:             <VariableRef name="PortCurrent"/>
182:         </DataMsg>
183:     </Notification>
184:     <Notification>
185:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
186:             <Qualifier name="telemetryLevel" value="1"/>
187:             <VariableRef name="Time" />
188:             <VariableRef name="SubS" />
189:             <VariableRef name="PortPowerState"/>
190:         </DataMsg>
191:     </Notification>
192: </Command>

```

```

193:         <CommandMsg id="3" name="ConfigureSoftTrip">
194:             <VariableRef name="SoftCurrentLimit"/>
195:         </CommandMsg>
196:     </Command>
197:     <Command>
198:         <CommandMsg id="4" name="PortPowerOn" />
199:     </Command>
200:     <Command>
201:         <CommandMsg id="5" name="PortPowerOff" />
202:     </Command>
203: </Interface>
204:
205:
206: <Interface id="6" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
207:     <Qualifier name="PortID" value="1"/>
208:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>
209:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
210:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
211:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
212:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
213:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
214:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
215:         <Drange name="PowerStateEnum">
216:             <Option name="Off" value="0"/>
217:             <Option name="On" value="1"/>
218:             <Option name="Tripped" value="2"/>
219:         </Drange>
220:     </Variable>
221:     <Notification>
222:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
223:             <Qualifier name="telemetryLevel" value="1"/>
224:             <VariableRef name="Time" />
225:             <VariableRef name="SubS" />
226:             <VariableRef name="PortPowerState" />
227:             <VariableRef name="PortVoltage"/>
228:             <VariableRef name="PortCurrent"/>
229:         </DataMsg>
230:     </Notification>
231: </Notification>

```

```

232:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
233:             <Qualifier name="telemetryLevel" value="1"/>
234:             <VariableRef name="Time" />
235:             <VariableRef name="SubS" />
236:             <VariableRef name="PortPowerState"/>
237:         </DataMsg>
238:     </Notification>
239:     <Command>
240:         <CommandMsg id="3" name="ConfigureSoftTrip">
241:             <VariableRef name="SoftCurrentLimit"/>
242:         </CommandMsg>
243:     </Command>
244:     <Command>
245:         <CommandMsg id="4" name="PortPowerOn" />
246:     </Command>
247:     <Command>
248:         <CommandMsg id="5" name="PortPowerOff" />
249:     </Command>
250: </Interface>
251:
252:
253: <Interface id="7" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
254:     <Qualifier name="PortID" value="2"/>
255:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>
256:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
257:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
258:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
259:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
260:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
261:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
262:         <Drange name="PowerStateEnum">
263:             <Option name="Off" value="0"/>
264:             <Option name="On" value="1"/>
265:             <Option name="Tripped" value="2"/>
266:         </Drange>
267:     </Variable>
268:     <Notification>
269:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
270:             <Qualifier name="telemetryLevel" value="1"/>

```

```

271:         <VariableRef name="Time" />
272:         <VariableRef name="SubS" />
273:         <VariableRef name="PortPowerState" />
274:         <VariableRef name="PortVoltage"/>
275:         <VariableRef name="PortCurrent"/>
276:     </DataMsg>
277: </Notification>
278: <Notification>
279:     <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
280:         <Qualifier name="telemetryLevel" value="1"/>
281:         <VariableRef name="Time" />
282:         <VariableRef name="SubS" />
283:         <VariableRef name="PortPowerState"/>
284:     </DataMsg>
285: </Notification>
286: <Command>
287:     <CommandMsg id="3" name="ConfigureSoftTrip">
288:         <VariableRef name="SoftCurrentLimit"/>
289:     </CommandMsg>
290: </Command>
291: <Command>
292:     <CommandMsg id="4" name="PortPowerOn" />
293: </Command>
294: <Command>
295:     <CommandMsg id="5" name="PortPowerOff" />
296: </Command>
297: </Interface>
298:
299:
300: <Interface id="8" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
301:     <Qualifier name="PortID" value="3"/>
302:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>
303:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
304:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
305:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
306:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
307:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
308:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
309:         <Drange name="PowerStateEnum">

```

```

310:         <Option name="Off" value="0"/>
311:         <Option name="On" value="1"/>
312:         <Option name="Tripped" value="2"/>
313:     </Drange>
314: </Variable>
315: <Notification>
316:     <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
317:         <Qualifier name="telemetryLevel" value="1"/>
318:         <VariableRef name="Time" />
319:         <VariableRef name="SubS" />
320:         <VariableRef name="PortPowerState" />
321:         <VariableRef name="PortVoltage"/>
322:         <VariableRef name="PortCurrent"/>
323:     </DataMsg>
324: </Notification>
325: <Notification>
326:     <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
327:         <Qualifier name="telemetryLevel" value="1"/>
328:         <VariableRef name="Time" />
329:         <VariableRef name="SubS" />
330:         <VariableRef name="PortPowerState"/>
331:     </DataMsg>
332: </Notification>
333: <Command>
334:     <CommandMsg id="3" name="ConfigureSoftTrip">
335:         <VariableRef name="SoftCurrentLimit"/>
336:     </CommandMsg>
337: </Command>
338: <Command>
339:     <CommandMsg id="4" name="PortPowerOn" />
340: </Command>
341: <Command>
342:     <CommandMsg id="5" name="PortPowerOff" />
343: </Command>
344: </Interface>
345:
346:
347: <Interface id="9" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
348:     <Qualifier name="PortID" value="4"/>
349:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>

```

```

350:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
351:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
352:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
353:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
354:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
355:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
356:         <Drange name="PowerStateEnum">
357:             <Option name="Off" value="0"/>
358:             <Option name="On" value="1"/>
359:             <Option name="Tripped" value="2"/>
360:         </Drange>
361:     </Variable>
362:     <Notification>
363:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
364:             <Qualifier name="telemetryLevel" value="1"/>
365:             <VariableRef name="Time" />
366:             <VariableRef name="SubS" />
367:             <VariableRef name="PortPowerState" />
368:             <VariableRef name="PortVoltage"/>
369:             <VariableRef name="PortCurrent"/>
370:         </DataMsg>
371:     </Notification>
372:     <Notification>
373:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
374:             <Qualifier name="telemetryLevel" value="1"/>
375:             <VariableRef name="Time" />
376:             <VariableRef name="SubS" />
377:             <VariableRef name="PortPowerState"/>
378:         </DataMsg>
379:     </Notification>
380:     <Command>
381:         <CommandMsg id="3" name="ConfigureSoftTrip">
382:             <VariableRef name="SoftCurrentLimit"/>
383:         </CommandMsg>
384:     </Command>
385:     <Command>
386:         <CommandMsg id="4" name="PortPowerOn" />
387:     </Command>
388:     <Command>

```



```

389:         <CommandMsg id="5" name="PortPowerOff" />
390:     </Command>
391: </Interface>
392:
393:
394: <Interface id="10" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
395:     <Qualifier name="PortID" value="5"/>
396:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>
397:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
398:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
399:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
400:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
401:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
402:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
403:         <Drange name="PowerStateEnum">
404:             <Option name="Off" value="0"/>
405:             <Option name="On" value="1"/>
406:             <Option name="Tripped" value="2"/>
407:         </Drange>
408:     </Variable>
409:     <Notification>
410:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
411:             <Qualifier name="telemetryLevel" value="1"/>
412:             <VariableRef name="Time" />
413:             <VariableRef name="SubS" />
414:             <VariableRef name="PortPowerState" />
415:             <VariableRef name="PortVoltage"/>
416:             <VariableRef name="PortCurrent"/>
417:         </DataMsg>
418:     </Notification>
419:     <Notification>
420:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
421:             <Qualifier name="telemetryLevel" value="1"/>
422:             <VariableRef name="Time" />
423:             <VariableRef name="SubS" />
424:             <VariableRef name="PortPowerState"/>
425:         </DataMsg>
426:     </Notification>
427: </Command>

```

```

428:         <CommandMsg id="3" name="ConfigureSoftTrip">
429:             <VariableRef name="SoftCurrentLimit"/>
430:         </CommandMsg>
431:     </Command>
432:     <Command>
433:         <CommandMsg id="4" name="PortPowerOn" />
434:     </Command>
435:     <Command>
436:         <CommandMsg id="5" name="PortPowerOff" />
437:     </Command>
438: </Interface>
439:
440:
441: <Interface id="11" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
442:     <Qualifier name="PortID" value="6"/>
443:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>
444:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
445:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
446:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
447:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
448:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
449:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
450:         <Drange name="PowerStateEnum">
451:             <Option name="Off" value="0"/>
452:             <Option name="On" value="1"/>
453:             <Option name="Tripped" value="2"/>
454:         </Drange>
455:     </Variable>
456:     <Notification>
457:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
458:             <Qualifier name="telemetryLevel" value="1"/>
459:             <VariableRef name="Time" />
460:             <VariableRef name="SubS" />
461:             <VariableRef name="PortPowerState" />
462:             <VariableRef name="PortVoltage"/>
463:             <VariableRef name="PortCurrent"/>
464:         </DataMsg>
465:     </Notification>
466: </Notification>

```

```

467:         <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
468:             <Qualifier name="telemetryLevel" value="1"/>
469:             <VariableRef name="Time" />
470:             <VariableRef name="SubS" />
471:             <VariableRef name="PortPowerState"/>
472:         </DataMsg>
473:     </Notification>
474:     <Command>
475:         <CommandMsg id="3" name="ConfigureSoftTrip">
476:             <VariableRef name="SoftCurrentLimit"/>
477:         </CommandMsg>
478:     </Command>
479:     <Command>
480:         <CommandMsg id="4" name="PortPowerOn" />
481:     </Command>
482:     <Command>
483:         <CommandMsg id="5" name="PortPowerOff" />
484:     </Command>
485: </Interface>
486:
487:
488: <Interface id="12" name="ASIMPortInterface" description="Interface for a single SPA-U port on
the hub.">
489:     <Qualifier name="PortID" value="7"/>
490:     <Qualifier name="BreakerTripCurrent" value="4.5" units="Amps"/>
491:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
492:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
493:     <Variable name="PortVoltage" kind="Voltage" format="FLOAT32" scaleUnits="Volts"/>
494:     <Variable name="PortCurrent" kind="Current" format="FLOAT32" scaleUnits="Amps"/>
495:     <Variable name="SoftCurrentLimit" kind="TripCurrent" format="FLOAT32"
defaultValue="1.0" scaleUnits="Amps"/>
496:     <Variable name="PortPowerState" kind="boolean" format="UINT08">
497:         <Drange name="PowerStateEnum">
498:             <Option name="Off" value="0"/>
499:             <Option name="On" value="1"/>
500:             <Option name="Tripped" value="2"/>
501:         </Drange>
502:     </Variable>
503:     <Notification>
504:         <DataMsg id="1" name="PortStatus" msgArrival="PERIODIC" msgRate="1" >
505:             <Qualifier name="telemetryLevel" value="1"/>

```

```

506:         <VariableRef name="Time" />
507:         <VariableRef name="SubS" />
508:         <VariableRef name="PortPowerState" />
509:         <VariableRef name="PortVoltage"/>
510:         <VariableRef name="PortCurrent"/>
511:     </DataMsg>
512: </Notification>
513: <Notification>
514:     <DataMsg id="2" name="PortTripped" msgArrival="EVENT">
515:         <Qualifier name="telemetryLevel" value="1"/>
516:         <VariableRef name="Time" />
517:         <VariableRef name="SubS" />
518:         <VariableRef name="PortPowerState"/>
519:     </DataMsg>
520: </Notification>
521: <Command>
522:     <CommandMsg id="3" name="ConfigureSoftTrip">
523:         <VariableRef name="SoftCurrentLimit"/>
524:     </CommandMsg>
525: </Command>
526: <Command>
527:     <CommandMsg id="4" name="PortPowerOn" />
528: </Command>
529: <Command>
530:     <CommandMsg id="5" name="PortPowerOff" />
531: </Command>
532: </Interface>
533:
534:
535: <Interface name="CmpSafety" id="13">
536:     <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
537:     <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
538:     <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
539:     <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
540:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
541:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
542:     <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32"
units="degC" />
543:     <Request>
544:         <CommandMsg name="GetDeviceTemperature" id="1" />

```

```

545:         <DataReplyMsg name="DeviceTempReply" id="2">
546:             <VariableRef name="Time" />
547:             <VariableRef name="SubS" />
548:             <VariableRef name="DeviceTemperature"/>
549:         </DataReplyMsg>
550:     </Request>
551:     <Notification>
552:         <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
553:             <Qualifier name="telemetryLevel" value="1"/>
554:             <VariableRef name="Time" />
555:             <VariableRef name="SubS" />
556:             <VariableRef name="DeviceTemperature"/>
557:         </DataMsg>
558:     </Notification>
559: </Interface>
560:
561: </xTEDS>

```

File: sdm/app/test/DMTTests/xTEDSRegTests/Thruster.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS                                xTEDS02.xsd"
name="MonoPropellantThrusterXTEDS"
4:   version="2.0">
5:   <Device    name="MonopropellantThruster"    kind="monoThrust"    description="A    single
monopropellant thruster" />
6:   <Interface name="ThrusterBasic" id="1">
7:     <Qualifier name="headID" value="0"/>
8:     <Variable    name="catBedReady"    format="UINT08"    kind="valid"    units="0_1"
description="Boolean byte indicating the 'ready to fire' status of the thruster">
9:       <Drange name="catBedReadyEnum">
10:         <Option name="notReady" value="0" />
11:         <Option name="ready" value="1" />
12:       </Drange>
13:     </Variable>
14:     <Command>
15:       <CommandMsg name="thrusterOnCmd" id="1" />
16:     </Command>
17:     <Command>
18:       <CommandMsg name="thrusterOffCmd" id="2" />
19:     </Command>
20:     <Request>
21:       <CommandMsg name="isReady" id="3" />
22:       <DataReplyMsg name="thrusterStatus" id="4">
23:         <VariableRef name="catBedReady" />
24:       </DataReplyMsg>
25:     </Request>
26:     <Request>
27:       <CommandMsg name="prepCatBed" id="4" />
28:       <DataReplyMsg name="readyToFire" id="5" />
29:     </Request>
30:   </Interface>
31:
32:   <Interface name="DevPwr" id="2">
33:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
34:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
35:     <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
36:     <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
```

```

37: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
38:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
39: <Variable name="DevPwrState" kind="Power_State" format="UINT08">
40:     <Drange name="DevPwrStateEnum">
41:         <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
42:         <Option name="DevPwrON" value="1" description="Device may draw full power." />
43:     </Drange>
44: </Variable>
45: <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
46:     <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
47:         <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
48:         <Option name="DevPwrON" value="1" description="Device may draw full power." />
49:     </Drange>
50: </Variable>
51: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
52: <Command>
53:     <CommandMsg name="DevPwrSetState" id="1">
54:         <VariableRef name="DevPwrStateSet" />
55:     </CommandMsg>
56:     <FaultMsg name="DevPwrStateNotSet" id="2">
57:         <VariableRef name="Time" />
58:         <VariableRef name="SubS" />
59:         <VariableRef name="DevPwrState" />
60:         <VariableRef name="DevPwrStateSet" />
61:     </FaultMsg>
62: </Command>
63: <Notification>
64:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
65:         <Qualifier name="telemetryLevel" value="1"/>
66:         <VariableRef name="Time" />
67:         <VariableRef name="SubS" />
68:         <VariableRef name="DevPwrState" />
69:         <VariableRef name="DevPwrStateSet" />
70:     </DataMsg>
71: </Notification>
72: <Request>
73:     <CommandMsg name="getPowerInMode" id="4" />
74:     <DataReplyMsg name="powerInMode" id="5">
75:         <VariableRef name="modePowers"/>

```

```

76:   </DataReplyMsg>
77: </Request>
78: </Interface>
79:
80: <Interface name="CmpSOH" id="3">
81:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
82:   <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
83:   <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
84:   <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
85:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
86:       <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
87:   <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
88: <Request>
89:   <CommandMsg name="GetDeviceTemperature" id="1" />
90:   <DataReplyMsg name="DeviceTempReply" id="2">
91:     <VariableRef name="Time" />
92:     <VariableRef name="SubS" />
93:     <VariableRef name="DeviceTemperature"/>
94:   </DataReplyMsg>
95: </Request>
96: <Notification>
97:   <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
98:     <Qualifier name="telemetryLevel" value="1"/>
99:     <VariableRef name="Time" />
100:    <VariableRef name="SubS" />
101:    <VariableRef name="DeviceTemperature"/>
102:   </DataMsg>
103: </Notification>
104: </Interface>
105: </xTEDS>

```


File: sdm/app/test/DMTests/xTEDSRegTests/RoboHub_lean.h

```
1: #ifndef _ROBOHUB_8_PORT_XTEDS_H
2: #define _ROBOHUB_8_PORT_XTEDS_H
3:
4: #define _STRING_ROBOHUB_8_PORT_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance      \"      xsi:schemaLocation=
\"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \" name= \"Robo_Hub_xTEDS
\" version= \"11.16\">\" \
7: "" \
8: "<Device name= \"RoboHub_8_Port\" kind= \"Robust_Hub\" modelId= \"0001\">\" \
9: "<Qualifier name= \"Manufacturer\" value= \"DataDesignCorp\"/>\" \
10: "<Qualifier name= \"Model\" value= \"Gen1\"/>\" \
11: "<Qualifier name= \"SerialNumber\" value= \"12345\"/>\" \
12: "</Device>\" \
13: "" \
14: "<!-- Panel interface -->\" \
15: "<Interface id= \"1\" name= \"PanelPowerInterface\" description= \"Panel power interface\">\" \
16: "<Qualifier name= \"NumberOfPorts\" value= \"4\"/>\" \
17: "<Qualifier name= \"BreakerTripCurrent\" value= \"30.0\" units= \"Amps\"/>\" \
18: "<Variable name= \"PortReference\" kind= \"PowerPortNumber\" format= \"UINT08\"/>\" \
19: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\"/>\" \
20: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\"
scaleFactor= \".0001\" scaleUnits= \"Seconds\"/>\" \
21: "<Variable name= \"PanelVoltage\" kind= \"Voltage\" format= \"INT16\" scaleFactor= \"0.1\"
scaleUnits= \"Volts\"/>\" \
22: "<Variable name= \"BreakerCurrentArray\" length= \"4\" kind= \"Current\" format= \"INT16\"
scaleFactor= \"1.0\" scaleUnits= \"Amps\">\" \
23: "<Qualifier name= \"Panel\" value= \"Array_4\"/>\" \
24: "</Variable>\" \
25: "<Variable name= \"PortPowerStateArray\" length= \"4\" kind= \"enumeration\" format= \"UINT08
\">\" \
26: "<Drange name= \"PowerStateEnum\">\" \
27: "<Option name= \"Open\" value= \"0\"/>\" \
28: "<Option name= \"Closed\" value= \"1\"/>\" \
29: "</Drange>\" \
30: "</Variable>\" \
31: "<Notification>\" \
32: "<DataMsg id= \"1\" name= \"PowerStatus\" msgArrival= \"PERIODIC\" msgRate= \"1\">\" \
33: "<Qualifier name= \"telemetryLevel\" value= \"1\"/>\" \
```

```

34: "<VariableRef name= \"Time \"/>" \
35: "<VariableRef name= \"SubS \"/>" \
36: "<VariableRef name= \"PanelVoltage \"/>" \
37: "<VariableRef name= \"BreakerCurrentArray \"/>" \
38: "<VariableRef name= \"PortPowerStateArray \"/>" \
39: "</DataMsg>" \
40: "</Notification>" \
41: "<Command>" \
42: "<CommandMsg id= \"2\" name= \"PortPowerOn \">" \
43: "<VariableRef name= \"PortReference \"/>" \
44: "</CommandMsg>" \
45: "</Command>" \
46: "<Command>" \
47: "<CommandMsg id= \"3\" name= \"PortPowerOff \">" \
48: "<VariableRef name= \"PortReference \"/>" \
49: "</CommandMsg>" \
50: "</Command>" \
51: "</Interface>" \
52: "" \
53: "<!-- Hub interface -->" \
54: "<Interface id= \"2\" name= \"RoboHubInterface \" description= \"Interface for robust hub \">" \
55: "<Qualifier name= \"NumberOfPorts \" value= \"8 \"/>" \
56: "<Qualifier name= \"PortTripCurrent \" value= \"4.5 \" units= \"Amps \"/>" \
57: "<Variable name= \"PortReference \" kind= \"HubPortNumber \" format= \"UINT08 \" description= \"Refers to a port for events and commands \"/>" \
58: "<Variable name= \"PortEnumeration \" kind= \"HubEnumerationStatus \" format= \"UINT08 \" description= \"Used to indicate hub enumeration state \"/>" \
59: "<Variable name= \"PPS_Status \" kind= \"PpsStatus \" format= \"UINT08 \" description= \"Used to indicate PPS status \"/>" \
60: "<Variable name= \"SetTripCurrentVal \" kind= \"TripCurrent \" format= \"INT16 \" scaleFactor= \"0.1 \" defaultValue= \"10 \" scaleUnits= \"Amps \"/>" \
61: "<Variable name= \"PortPowerState \" kind= \"PortPowerState \" format= \"UINT08 \">" \
62: "<Drange name= \"PowerStateEnum \">" \
63: "<Option name= \"Open \" value= \"0 \"/>" \
64: "<Option name= \"Closed \" value= \"1 \"/>" \
65: "<Option name= \"HardTrip \" value= \"2 \"/>" \
66: "<Option name= \"SoftTrip \" value= \"3 \"/>" \
67: "</Drange>" \
68: "</Variable>" \
69: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />" \
70: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />" \

```

```

71: "<Variable name= \"PortVoltageArray \" length= \"8 \" kind= \"Voltage \" format= \"INT16 \"
scaleFactor= \"0.1 \" scaleUnits= \"Volts \"/>\" \
72: "<Qualifier name= \"HubPort \" value= \"Array_8 \"/>\" \
73: "</Variable>\" \
74: "<Variable name= \"PortCurrentArray \" length= \"8 \" kind= \"Current \" format= \"INT16 \"
scaleFactor= \"0.1 \" scaleUnits= \"Amps \"/>\" \
75: "<Qualifier name= \"HubPort \" value= \"Array_8 \"/>\" \
76: "</Variable>\" \
77: "<Variable name= \"SoftCurrentLimitArray \" length= \"8 \" kind= \"TripCurrent \" format= \"INT16
\" scaleFactor= \"0.1 \" defaultValue= \"10 \" scaleUnits= \"Amps \"/>\" \
78: "<Qualifier name= \"HubPort \" value= \"Array_8 \"/>\" \
79: "</Variable>\" \
80: "<Variable name= \"PortPowerStateArray \" length= \"8 \" kind= \"boolean \" format= \"UINT08 \"/>\"
\
81: "<Qualifier name= \"HubPort \" value= \"Array_8 \"/>\" \
82: "<Drange name= \"PowerStateEnumArray \"/>\" \
83: "<Option name= \"Open \" value= \"0 \"/>\" \
84: "<Option name= \"Closed \" value= \"1 \"/>\" \
85: "<Option name= \"HardTrip \" value= \"2 \"/>\" \
86: "<Option name= \"SoftTrip \" value= \"3 \"/>\" \
87: "</Drange>\" \
88: "</Variable>\" \
89: "<Notification>\" \
90: "<DataMsg id= \"1 \" name= \"PortStatus \" msgArrival= \"PERIODIC \" msgRate= \"1 \"/>\" \
91: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
92: "<VariableRef name= \"Time \"/>\" \
93: "<VariableRef name= \"SubS \"/>\" \
94: "<VariableRef name= \"PortPowerStateArray \"/>\" \
95: "<VariableRef name= \"PortVoltageArray \"/>\" \
96: "<VariableRef name= \"PortCurrentArray \"/>\" \
97: "</DataMsg>\" \
98: "</Notification>\" \
99: "<Notification>\" \
100: "<DataMsg id= \"2 \" name= \"PortTripped \" msgArrival= \"EVENT \"/>\" \
101: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
102: "<VariableRef name= \"Time \"/>\" \
103: "<VariableRef name= \"SubS \"/>\" \
104: "<VariableRef name= \"PortReference \"/>\" \
105: "<VariableRef name= \"PortPowerState \"/>\" \
106: "</DataMsg>\" \
107: "</Notification>\" \
108: "<Command>\" \

```

```

109: "<CommandMsg id= \"3\" name= \"ConfigureSoftTrip \"/>" \
110: "<VariableRef name= \"PortReference \"/>" \
111: "<VariableRef name= \"SetTripCurrentVal \"/>" \
112: "</CommandMsg>" \
113: "</Command>" \
114: "<Command>" \
115: "<CommandMsg id= \"4\" name= \"PortPowerOn \">" \
116: "<VariableRef name= \"PortReference \"/>" \
117: "</CommandMsg>" \
118: "</Command>" \
119: "<Command>" \
120: "<CommandMsg id= \"5\" name= \"PortPowerOff \">" \
121: "<VariableRef name= \"PortReference \"/>" \
122: "</CommandMsg>" \
123: "</Command>" \
124: "<Request>" \
125: "<CommandMsg id= \"6\" name= \"GetHubStatus \"/>" \
126: "<DataReplyMsg id= \"7\" name= \"HubStatusReply \"/>" \
127: "<VariableRef name= \"Time \"/>" \
128: "<VariableRef name= \"SubS \"/>" \
129: "<VariableRef name= \"PortEnumeration \"/>" \
130: "<VariableRef name= \"PPS_Status \"/>" \
131: "</DataReplyMsg>" \
132: "</Request>" \
133: "<Request>" \
134: "<CommandMsg id= \"8\" name= \"GetSoftTripLimits \"/>" \
135: "<DataReplyMsg id= \"9\" name= \"SoftLimitSettingsReply \"/>" \
136: "<VariableRef name= \"Time \"/>" \
137: "<VariableRef name= \"SubS \"/>" \
138: "<VariableRef name= \"SoftCurrentLimitArray \"/>" \
139: "</DataReplyMsg>" \
140: "</Request>" \
141: "</Interface>" \
142: "" \
143: "<!-- Component safety interface -->" \
144: "<Interface name= \"CmpSafety\" id= \"3\">" \
145: "<Qualifier name= \"TemperatureLoKeepout\" value= \"-20.0\" units= \"degC \"/>" \
146: "<Qualifier name= \"TemperatureLoWarning\" value= \"-10.0\" units= \"degC \"/>" \
147: "<Qualifier name= \"TemperatureHiWarning\" value= \"50.0\" units= \"degC \"/>" \
148: "<Qualifier name= \"TemperatureHiKeepout\" value= \"60.0\" units= \"degC \"/>" \
149: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds \"/>" \

```

```

150: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \"/>" \
151: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"INT16 \" scaleFactor=
\"1.0\" scaleUnits= \"degC \"/>" \
152: "<Variable name= \"PanelTemperatureArray \" length= \"8 \" kind= \"temperature \" format=
\"INT16\" scaleFactor= \"1.0\" scaleUnits= \"degC \"/>" \
153: "<Qualifier name= \"PanelTemperatureChannel \" value= \"Array_8 \"/>" \
154: "</Variable>" \
155: "<Notification>" \
156: "<DataMsg name= \"DeviceTemp \" id= \"1 \" msgArrival= \"PERIODIC \" msgRate= \"1 \"/>" \
157: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
158: "<VariableRef name= \"Time \"/>" \
159: "<VariableRef name= \"SubS \"/>" \
160: "<VariableRef name= \"DeviceTemperature \"/>" \
161: "<VariableRef name= \"PanelTemperatureArray \"/>" \
162: "</DataMsg>" \
163: "</Notification>" \
164: "</Interface>" \
165: "" \
166: "</xTEDS>" \
167: ""
168:
169: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/DownlinkController.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS    ../Schema/xTEDS02.xsd"
name="DownlinkControllerXTEDS" description="DownlinkController xTEDS" version="1.0">
4:
5:         <Application    name="DownlinkController"    kind="CommunicationsFlightSoftware"
description="Communications Subsystem Uplinked Command Controller"/>
6:
7: <Interface name="DownlinkControllerInterface" id="1" description="Basic interface for scheduling
activities and updating their status">
8:
9: <Variable name="FileID" kind="tbd" format="UINT08" length="256"/>
10:
11: <Command>
12: <CommandMsg name="DownlinkTelemetry" id="002" description="Deliver a unit of telemetry
to the downlink"/>
13: </Command>
14:
15: <Command>
16: <CommandMsg name="DownlinkFile" id="003" description="Prep and begin downlinking the
referenced file">
17: <VariableRef name="FileID"/>
18: </CommandMsg>
19: </Command>
20:
21: <Command>
22: <CommandMsg name="DownlinkSSOH" id="004" description="Prep and begin downlinking
Stored State of Health"/>
23: </Command>
24:
25: </Interface>
26:
27: <Interface name="DownlinkControllerStatusIf" id="2">
28:
29: <Variable name="DownlinkControllerStatus" kind="Status" format="UINT08">
30: <Drange name="DownlinkControllerStatusEnum">
31: <Option value="0" name="NOT_INITIALIZED"/>
32: <Option value="1" name="INITIALIZING"/>
33: <Option value="1" name="RUNNING"/>
34: <Option value="2" name="TERMINATING"/>
```

```

35:     </Drange>
36: </Variable>
37:
38: <Variable name="DataMsgsDownlinked"      kind="tbd"          format="UINT32"/>
39: <Variable name="DataRequestsReceived"     kind="tbd"          format="UINT32"/>
40: <Variable name="DataMsgsInBuffer"         kind="tbd"          format="UINT32"/>
41: <Variable name="ComponentsRegistered"     kind="tbd"          format="UINT16"/>
42: <Variable name="FilesRequested"           kind="tbd"          format="UINT16"/>
43: <Variable name="FilesSuccessfullyDownlinked" kind="tbd"          format="UINT16"/>
44: <Variable name="FilesFailedDownlinked"    kind="tbd"          format="UINT16"/>
45: <Variable name="CommandsAccepted"         kind="tbd"          format="UINT16"/>
46: <Variable name="CommandsRejected"         kind="tbd"          format="UINT16"/>
47:
48: <Notification>
49:   <DataMsg name="DownlinkControllerStatusMsg" id="001" msgArrival="EVENT">
50:     <Qualifier value="1" name="telemetryLevel"/>
51:     <VariableRef name="DownlinkControllerStatus"/>
52:     <VariableRef name="ComponentsRegistered"/>
53:     <VariableRef name="DataMsgsDownlinked"/>
54:     <VariableRef name="DataRequestsReceived"/>
55:     <VariableRef name="DataMsgsInBuffer"/>
56:     <VariableRef name="CommandsAccepted"/>
57:     <VariableRef name="CommandsRejected"/>
58:     <VariableRef name="FilesRequested"/>
59:     <VariableRef name="FilesSuccessfullyDownlinked"/>
60:     <VariableRef name="FilesFailedDownlinked"/>
61:   </DataMsg>
62: </Notification>
63:
64: </Interface>
65:
66: <Interface name="ICSDebugInterface" id="3">
67:
68:   <!--
69:   Note: DebugLevel is a bit field with the following assigned values:
70:   DEBUG_ENTRY_AND_EXIT   = 0x01,
71:   DEBUG_ENTRY_PARAMETERS = 0x02,
72:   DEBUG_EXIT_PARAMETERS  = 0x04,
73:   DEBUG_LEVEL_LOW       = 0x08,
74:   DEBUG_LEVEL_MEDIUM    = 0x10,
75:   DEBUG_LEVEL_HIGH      = 0x20.

```

```

76:   The values are OR'd to determine the debug information to be logged.
77:   -->
78:
79:   <Variable name="DebugLevel" kind="tbd" format="UINT16"/>
80:   <Variable name="CurrentDebugLevel" kind="tbd" format="UINT16"/>
81:
82:   <Variable name="SetDebugLevelReceived" kind="tbd" format="UINT16"/>
83:   <Variable name="SetDebugLevelAccepted" kind="tbd" format="UINT16"/>
84:   <Variable name="SetDebugLevelSuccess" kind="tbd" format="UINT16"/>
85:   <Variable name="SetDebugLevelFailure" kind="tbd" format="UINT16"/>
86:
87:   <Command>
88:       <CommandMsg name="SetDebugLevel" id="001" description="Set the debug log verbosity
level">
89:           <VariableRef name="DebugLevel"/>
90:       </CommandMsg>
91:   </Command>
92:
93:   <Command>
94:       <CommandMsg name="SendDebugStatus" id="002"/>
95:   </Command>
96:
97:   <Notification>
98:       <DataMsg name="DebugStatus" id="003" msgArrival="EVENT">
99:           <Qualifier value="1" name="telemetryLevel"/>
100:          <VariableRef name="CurrentDebugLevel"/>
101:          <VariableRef name="SetDebugLevelReceived"/>
102:          <VariableRef name="SetDebugLevelAccepted"/>
103:          <VariableRef name="SetDebugLevelSuccess"/>
104:          <VariableRef name="SetDebugLevelFailure"/>
105:      </DataMsg>
106:  </Notification>
107:
108: </Interface>
109:
110: <Interface name="ICSTaskControlInterface" id="4">
111:
112:   <Command>
113:       <CommandMsg name="DestroyTask" id="001"/>
114:   </Command>
115:

```


116: <!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->
117: <!-- Other possible requests include GetPriority, and GetHeartBeatCount -->
118:
119: </Interface>
120:
121: </xTEDS>

File: sdm/app/test/DMTests/xTEDSRegTests/IRU.h

```
1: #ifndef _LN200S_IRU_XTEDS_H
2: #define _LN200S_IRU_XTEDS_H
3:
4: #define _STRING_LN200S_IRU_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"IRUxTeds \"\" \
8: "version= \"2.0\">\" \
9: "<Device name= \"LN200s_IRU\" kind= \"iru\" description= \"Litton LN200s IRU\" />\" \
10: "" \
11: "<Interface name= \"IRUBasic\" id= \"1\">\" \
12: "<Qualifier name= \"headID\" value= \"0\" />\" \
13: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />\" \
14: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\"
scaleFactor= \".0001\" scaleUnits= \"Seconds\" />\" \
15: "<Variable name= \"AngularRate\" kind= \"attitudeRate\" units= \"rad_s\" format= \"FLOAT32\"
length= \"3\" description= \"Angular rates about each primary axis of the IRU\">\" \
16: "<Qualifier name= \"representation\" value= \"vector\" />\" \
17: "<Qualifier name= \"frameMeasured\" value= \"DVF\" />\" \
18: "<Qualifier name= \"frameResolved\" value= \"DVF\" />\" \
19: "</Variable>\" \
20: "<Notification>\" \
21: "<DataMsg name= \"AnglRate\" msgArrival= \"PERIODIC\" msgRate= \"1\" id= \"1\">\" \
22: "<Qualifier name= \"telemetryLevel\" value= \"1\" />\" \
23: "<VariableRef name= \"Time\" />\" \
24: "<VariableRef name= \"SubS\" />\" \
25: "<VariableRef name= \"AngularRate\" />\" \
26: "</DataMsg>\" \
27: "</Notification>\" \
28: "</Interface>\" \
29: "" \
30: "<Interface name= \"DevPwr\" id= \"2\">\" \
31: "<Qualifier name= \"CurrentLoKeepout\" value= \"0.0\" units= \"A\" />\" \
32: "<Qualifier name= \"CurrentLoWarning\" value= \"0.0\" units= \"A\" />\" \
33: "<Qualifier name= \"CurrentHiWarning\" value= \"3.5\" units= \"A\" />\" \
34: "<Qualifier name= \"CurrentHiKeepout\" value= \"3.5\" units= \"A\" />\" \
35: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds\" />\" \
```

```

36: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001\" scaleUnits= \"Seconds \" />" \
37: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \">" \
38: "<Drange name= \"DevPwrStateEnum \">" \
39: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />"
\
40: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
41: "</Drange>" \
42: "</Variable>" \
43: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \">" \
44: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to
DevPwrStateEnumeration. \">" \
45: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \" />"
\
46: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />" \
47: "</Drange>" \
48: "</Variable>" \
49: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length=
\"2 \" />" \
50: "<Command>" \
51: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \">" \
52: "<VariableRef name= \"DevPwrStateSet \" />" \
53: "</CommandMsg>" \
54: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \">" \
55: "<VariableRef name= \"Time \" />" \
56: "<VariableRef name= \"SubS \" />" \
57: "<VariableRef name= \"DevPwrState \" />" \
58: "<VariableRef name= \"DevPwrStateSet \" />" \
59: "</FaultMsg>" \
60: "</Command>" \
61: "<Notification>" \
62: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \">" \
63: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />" \
64: "<VariableRef name= \"Time \" />" \
65: "<VariableRef name= \"SubS \" />" \
66: "<VariableRef name= \"DevPwrState \" />" \
67: "<VariableRef name= \"DevPwrStateSet \" />" \
68: "</DataMsg>" \
69: "</Notification>" \
70: "<Request>" \
71: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />" \
72: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">" \

```

```

73: "<VariableRef name= \"modePowers \"/>" \
74: "</DataReplyMsg>" \
75: "</Request>" \
76: "</Interface>" \
77: "" \
78: "<Interface name= \"CmpSOH \" id= \"3 \"/>" \
79: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>" \
80: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>" \
81: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>" \
82: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>" \
83: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>" \
84: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \" scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>" \
85: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units= \"degC \"/>" \
86: "<Request>" \
87: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \"/>" \
88: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \"/>" \
89: "<VariableRef name= \"Time \"/>" \
90: "<VariableRef name= \"SubS \"/>" \
91: "<VariableRef name= \"DeviceTemperature \"/>" \
92: "</DataReplyMsg>" \
93: "</Request>" \
94: "<Notification>" \
95: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \"/>" \
96: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
97: "<VariableRef name= \"Time \"/>" \
98: "<VariableRef name= \"SubS \"/>" \
99: "<VariableRef name= \"DeviceTemperature \"/>" \
100: "</DataMsg>" \
101: "</Notification>" \
102: "</Interface>" \
103: "</xTEDS>" \
104: ""
105:
106: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/RWheelAssy.h

```
1: #ifndef _REACTIONWHEELASSEMBLY_XTEDS_H
2: #define _REACTIONWHEELASSEMBLY_XTEDS_H
3:
4: #define _STRING_REACTIONWHEELASSEMBLY_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"utf-8\" ?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   "xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
\"ReactionWheelAssemblXxTEDS \"\" \
8:   "version= \"2.0\">\" \
9:   "<Device name= \"ReactionWheelAssembly \" kind= \"rwa \" description= \"xTeds for a 3-axis
reaction wheel assembly \" />\" \
10:   "" \
11:   "<!--The assembly can be commanded as a unit, or each wheel can be commanded individually-->\" \
12:   "<!--The first interface provides the aggregate behavior-->\" \
13:   "<Interface name= \"RWAssyInterface \" id= \"1 \">\" \
14:   "<Qualifier name= \"headID \" value= \"0 \" />\" \
15:   "<Variable name= \"torqueCommanded \" kind= \"torque \" format= \"FLOAT32 \" length= \"3 \"
units= \"Nm \" description= \"Torque vector to be applied to the spacecraft \">\" \
16:   "<Qualifier name= \"Representation \" value= \"vector \" />\" \
17:   "<Qualifier name= \"Frame_Measured \" value= \"DVF \" />\" \
18:   "<Qualifier name= \"Frame_Resolved \" value= \"DVF \" />\" \
19:   "</Variable>\" \
20:   "<Command>\" \
21:   "<!-- note: the name of a command uniquely identifies it within the dictionary -->\" \
22:   "<CommandMsg name= \"TorqueVectorCmd \" id= \"1 \" description= \"A 3D torque vector
command sent to the assembly as a whole \">\" \
23:   "<VariableRef name= \"torqueCommanded \" />\" \
24:   "</CommandMsg>\" \
25:   "</Command>\" \
26:   "</Interface>\" \
27:   "" \
28:   "<!--The assembly also exposes each wheel as if it were an individual component-->\" \
29:   "<!--The first (X-Axis) wheel-->\" \
30:   "<Interface name= \"RWSingleInterface \" id= \"2 \">\" \
31:   "<Qualifier name= \"headID \" value= \"1 \" />\" \
32:   "<Qualifier name= \"MaxMomentum \" value= \"5.0 \" units= \"Nms \" />\" \
33:   "<Qualifier name= \"NominalMomentum \" value= \"2.0 \" units= \"Nms \" />\" \
34:   "<Qualifier name= \"TorqueLossMomentum \" value= \"1.0 \" units= \"Nms \" />\" \
35:   "<Qualifier name= \"MaxTorqueAtMaxSpeed \" value= \"10.0 \" units= \"Nm \" />\" \
```

```

36: "<Qualifier name= \"MaxTorqueAtNominalSpeed \" value= \"5.0 \" units= \"Nm \"/>\" \
37: "<Qualifier name= \"TimeConstant \" value= \"0.1 \" units= \"s \"/>\" \
38: "<Qualifier name= \"IdlePower \" value= \"1.0 \" units= \"W \"/>\" \
39: "<Qualifier name= \"MaxPower \" value= \"10.0 \" units= \"W \"/>\" \
40: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>\" \
41: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>\" \
42: "<Variable name= \"opMode \" kind= \"mode \" format= \"UINT16 \">\" \
43: "<Drange name= \"enumModes \">\" \
44: "<Option name= \"idle \" value= \"0 \"/>\" \
45: "<Option name= \"operating \" value= \"1 \"/>\" \
46: "</Drange>\" \
47: "</Variable>\" \
48: "<Variable name= \"dataQuality \" kind= \"DataQuality \" format= \"UINT08 \">\" \
49: "<Drange name= \"DataQualityEnum \">\" \
50: "<Option name= \"dataBad \" value= \"0 \" description= \"data is garbage or NaN. \" />\" \
51: "<Option name= \"dataPoor \" value= \"1 \" description= \"data quality is poor. \" />\" \
52: "<Option name= \"dataGood \" value= \"2 \" description= \"data is good. \" />\" \
53: "</Drange>\" \
54: "</Variable>\" \
55: "<Variable name= \"commandedTorque \" kind= \"torque \" format= \"FLOAT32 \" units= \"Nm \"
description= \"Magnitude of torque requested of this device \" />\" \
56: "<Variable name= \"currentMomentum \" kind= \"angularMomentum \" format= \"FLOAT32 \"
units= \"Nms \" description= \"The current momentum of this wheel. \" />\" \
57: "<Variable name= \"satPercentage \" kind= \"saturationLevel \" format= \"FLOAT32 \" units=
\"percent \" description= \"The current percentage of saturation of this wheel. \" />\" \
58: "<Command>\" \
59: "<CommandMsg name= \"SingleAxisTorqueCmd \" id= \"1 \">\" \
60: "<VariableRef name= \"commandedTorque \" />\" \
61: "</CommandMsg>\" \
62: "</Command>\" \
63: "" \
64: "<Notification>\" \
65: "<DataMsg name= \"wheelSatLevel \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"2 \">\" \
66: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
67: "<VariableRef name= \"Time \" />\" \
68: "<VariableRef name= \"SubS \" />\" \
69: "<VariableRef name= \"dataQuality \" />\" \
70: "<VariableRef name= \"satPercentage \" />\" \
71: "</DataMsg>\" \
72: "</Notification>\" \
73: "<Notification>\" \

```

```

74: "<DataMsg name= \"MomentumCurrent \" msgArrival= \"PERIODIC \" msgRate= \"10 \" id= \"3
\">\" \
75: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
76: "<VariableRef name= \"Time \" />\" \
77: "<VariableRef name= \"SubS \" />\" \
78: "<VariableRef name= \"dataQuality \" />\" \
79: "<VariableRef name= \"currentMomentum \" />\" \
80: "</DataMsg>\" \
81: "</Notification>\" \
82: "<Command>\" \
83: "<CommandMsg name= \"setOpMode \" id= \"4 \">\" \
84: "<VariableRef name= \"opMode \"/>\" \
85: "</CommandMsg>\" \
86: "</Command>\" \
87: "<Notification>\" \
88: "<DataMsg name= \"opModeChanged \" id= \"5 \" msgArrival= \"EVENT \">\" \
89: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \
90: "<VariableRef name= \"opMode \"/>\" \
91: "</DataMsg>\" \
92: "</Notification>\" \
93: "</Interface>\" \
94: "" \
95: "<!--The 2nd (Y-Axis) wheel-->\" \
96: "<Interface name= \"RWSingleInterface \" id= \"3 \">\" \
97: "<Qualifier name= \"headID \" value= \"2 \"/>\" \
98: "<Qualifier name= \"MaxMomentum \" value= \"5.0 \" units= \"Nms \"/>\" \
99: "<Qualifier name= \"NominalMomentum \" value= \"2.0 \" units= \"Nms \"/>\" \
100: "<Qualifier name= \"TorqueLossMomentum \" value= \"1.0 \" units= \"Nms \"/>\" \
101: "<Qualifier name= \"MaxTorqueAtMaxSpeed \" value= \"10.0 \" units= \"Nm \"/>\" \
102: "<Qualifier name= \"MaxTorqueAtNominalSpeed \" value= \"5.0 \" units= \"Nm \"/>\" \
103: "<Qualifier name= \"TimeConstant \" value= \"0.1 \" units= \"s \"/>\" \
104: "<Qualifier name= \"IdlePower \" value= \"1.0 \" units= \"W \"/>\" \
105: "<Qualifier name= \"MaxPower \" value= \"10.0 \" units= \"W \"/>\" \
106: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \"/>\" \
107: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \"/>\" \
108: "<Variable name= \"opMode \" kind= \"mode \" format= \"UINT16 \">\" \
109: "<Drange name= \"enumModes \">\" \
110: "<Option name= \"idle \" value= \"0 \"/>\" \
111: "<Option name= \"operating \" value= \"1 \"/>\" \
112: "</Drange>\" \

```

```

113: "</Variable>" \
114: "<Variable name= \"dataQuality \" kind= \"DataQuality \" format= \"UINT08 \">" \
115: "<Drange name= \"DataQualityEnum \">" \
116: "<Option name= \"dataBad \" value= \"0 \" description= \"data is garbage or NaN. \" />" \
117: "<Option name= \"dataPoor \" value= \"1 \" description= \"data quality is poor. \" />" \
118: "<Option name= \"dataGood \" value= \"2 \" description= \"data is good. \" />" \
119: "</Drange>" \
120: "</Variable>" \
121: "<Variable name= \"commandedTorque \" kind= \"torque \" format= \"FLOAT32 \" units= \"Nm \"
description= \"Magnitude of torque requested of this device \" />" \
122: "<Variable name= \"currentMomentum \" kind= \"angularMomentum \" format= \"FLOAT32 \"
units= \"Nms \" description= \"The current momentum of this wheel. \" />" \
123: "<Variable name= \"satPercentage \" kind= \"saturationLevel \" format= \"FLOAT32 \" units=
\"percent \" description= \"The current percentage of saturation of this wheel. \" />" \
124: "<Command>" \
125: "<CommandMsg name= \"SingleAxisTorqueCmd \" id= \"1 \">" \
126: "<VariableRef name= \"commandedTorque \" />" \
127: "</CommandMsg>" \
128: "</Command>" \
129: "<Notification>" \
130: "<DataMsg name= \"wheelSatLevel \" msgArrival= \"PERIODIC \" msgRate= \"1 \" id= \"2 \">" \
131: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
132: "<VariableRef name= \"Time \" />" \
133: "<VariableRef name= \"SubS \" />" \
134: "<VariableRef name= \"dataQuality \" />" \
135: "<VariableRef name= \"satPercentage \" />" \
136: "</DataMsg>" \
137: "</Notification>" \
138: "<Notification>" \
139: "<DataMsg name= \"MomentumCurrent \" msgArrival= \"PERIODIC \" msgRate= \"10 \" id= \"3
\">" \
140: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>" \
141: "<VariableRef name= \"Time \" />" \
142: "<VariableRef name= \"SubS \" />" \
143: "<VariableRef name= \"dataQuality \" />" \
144: "<VariableRef name= \"currentMomentum \" />" \
145: "</DataMsg>" \
146: "</Notification>" \
147: "<Command>" \
148: "<CommandMsg name= \"setOpMode \" id= \"4 \">" \
149: "<VariableRef name= \"opMode \" />" \
150: "</CommandMsg>" \

```



```

151: "</Command>" \
152: "<Notification>" \
153: "<DataMsg name= \"opModeChanged\" id= \"5\" msgArrival= \"EVENT \"/>" \
154: "<Qualifier name= \"telemetryLevel\" value= \"1 \"/>" \
155: "<VariableRef name= \"opMode \"/>" \
156: "</DataMsg>" \
157: "</Notification>" \
158: "</Interface>" \
159: "" \
160: "<!--The 3rd (Z-Axis) wheel-->" \
161: "<Interface name= \"RWSingleInterface\" id= \"4 \"/>" \
162: "<Qualifier name= \"headID\" value= \"3 \"/>" \
163: "<Qualifier name= \"MaxMomentum\" value= \"5.0\" units= \"Nms \"/>" \
164: "<Qualifier name= \"NominalMomentum\" value= \"2.0\" units= \"Nms \"/>" \
165: "<Qualifier name= \"TorqueLossMomentum\" value= \"1.0\" units= \"Nms \"/>" \
166: "<Qualifier name= \"MaxTorqueAtMaxSpeed\" value= \"10.0\" units= \"Nm \"/>" \
167: "<Qualifier name= \"MaxTorqueAtNominalSpeed\" value= \"5.0\" units= \"Nm \"/>" \
168: "<Qualifier name= \"TimeConstant\" value= \"0.1\" units= \"s \"/>" \
169: "<Qualifier name= \"IdlePower\" value= \"1.0\" units= \"W \"/>" \
170: "<Qualifier name= \"MaxPower\" value= \"10.0\" units= \"W \"/>" \
171: "<Variable kind= \"Time\" name= \"Time\" format= \"UINT32\" units= \"Seconds \"/>" \
172: "<Variable kind= \"SubSeconds\" name= \"SubS\" units= \"Counts\" format= \"UINT32\" \
scaleFactor= \".0001\" scaleUnits= \"Seconds \"/>" \
173: "<Variable name= \"opMode\" kind= \"mode\" format= \"UINT16\" >" \
174: "<Drange name= \"enumModes\" >" \
175: "<Option name= \"idle\" value= \"0 \"/>" \
176: "<Option name= \"operating\" value= \"1 \"/>" \
177: "</Drange>" \
178: "</Variable>" \
179: "<Variable name= \"dataQuality\" kind= \"DataQuality\" format= \"UINT08 \"/>" \
180: "<Drange name= \"DataQualityEnum\" >" \
181: "<Option name= \"dataBad\" value= \"0\" description= \"data is garbage or NaN. \"/>" \
182: "<Option name= \"dataPoor\" value= \"1\" description= \"data quality is poor. \"/>" \
183: "<Option name= \"dataGood\" value= \"2\" description= \"data is good. \"/>" \
184: "</Drange>" \
185: "</Variable>" \
186: "<Variable name= \"commandedTorque\" kind= \"torque\" format= \"FLOAT32\" units= \"Nm\" \
description= \"Magnitude of torque requested of this device \"/>" \
187: "<Variable name= \"currentMomentum\" kind= \"angularMomentum\" format= \"FLOAT32\" \
units= \"Nms\" description= \"The current momentum of this wheel. \"/>" \
188: "<Variable name= \"satPercentage\" kind= \"saturationLevel\" format= \"FLOAT32\" units= \
\"percent\" description= \"The current percentage of saturation of this wheel. \"/>" \

```

189: "<Command>" \
 190: "<CommandMsg name= \"SingleAxisTorqueCmd\" id= \"1 \">" \
 191: "<VariableRef name= \"commandedTorque\" />" \
 192: "</CommandMsg>" \
 193: "</Command>" \
 194: "<Notification>" \
 195: "<DataMsg name= \"wheelSatLevel\" msgArrival= \"PERIODIC\" msgRate= \"1\" id= \"2 \">" \
 196: "<Qualifier name= \"telemetryLevel\" value= \"1\" />" \
 197: "<VariableRef name= \"Time\" />" \
 198: "<VariableRef name= \"SubS\" />" \
 199: "<VariableRef name= \"dataQuality\" />" \
 200: "<VariableRef name= \"satPercentage\" />" \
 201: "</DataMsg>" \
 202: "</Notification>" \
 203: "<Notification>" \
 204: "<DataMsg name= \"MomentumCurrent\" msgArrival= \"PERIODIC\" msgRate= \"10\" id= \"3 \">" \
 205: "<Qualifier name= \"telemetryLevel\" value= \"1\" />" \
 206: "<VariableRef name= \"Time\" />" \
 207: "<VariableRef name= \"SubS\" />" \
 208: "<VariableRef name= \"dataQuality\" />" \
 209: "<VariableRef name= \"currentMomentum\" />" \
 210: "</DataMsg>" \
 211: "</Notification>" \
 212: "<Command>" \
 213: "<CommandMsg name= \"setOpMode\" id= \"4 \">" \
 214: "<VariableRef name= \"opMode\" />" \
 215: "</CommandMsg>" \
 216: "</Command>" \
 217: "<Notification>" \
 218: "<DataMsg name= \"opModeChanged\" id= \"5\" msgArrival= \"EVENT\" />" \
 219: "<Qualifier name= \"telemetryLevel\" value= \"1\" />" \
 220: "<VariableRef name= \"opMode\" />" \
 221: "</DataMsg>" \
 222: "</Notification>" \
 223: "</Interface>" \
 224: "" \
 225: "<Interface name= \"DevPwr\" id= \"5 \">" \
 226: "<Qualifier name= \"CurrentLoKeepout\" value= \"0.0\" units= \"A\" />" \
 227: "<Qualifier name= \"CurrentLoWarning\" value= \"0.0\" units= \"A\" />" \
 228: "<Qualifier name= \"CurrentHiWarning\" value= \"3.5\" units= \"A\" />" \

229: "<Qualifier name= \"CurrentHiKeepout \" value= \"3.5 \" units= \"A \"/>\" \\
 230: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \\
 231: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
 scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \\
 232: "<Variable name= \"DevPwrState \" kind= \"Power_State \" format= \"UINT08 \"/>\" \\
 233: "<Drange name= \"DevPwrStateEnum \"/>\" \\
 234: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \"
 />\" \\
 235: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />\" \\
 236: "</Drange>\" \\
 237: "</Variable>\" \\
 238: "<Variable name= \"DevPwrStateSet \" kind= \"Power_State \" format= \"UINT08 \" id= \"2 \"/>\" \\
 239: "<Drange name= \"DevPwrStateEnumReference \" description= \"This should be a reference to
 DevPwrStateEnumeration. \"/>\" \\
 240: "<Option name= \"DevPwrOFF \" value= \"0 \" description= \"All power to device is turned off. \"
 />\" \\
 241: "<Option name= \"DevPwrON \" value= \"1 \" description= \"Device may draw full power. \" />\" \\
 242: "</Drange>\" \\
 243: "</Variable>\" \\
 244: "<Variable name= \"modePowers \" kind= \"power \" format= \"FLOAT32 \" units= \"W \" length=
 \"2 \" />\" \\
 245: "<Command>\" \\
 246: "<CommandMsg name= \"DevPwrSetState \" id= \"1 \"/>\" \\
 247: "<VariableRef name= \"DevPwrStateSet \" />\" \\
 248: "</CommandMsg>\" \\
 249: "<FaultMsg name= \"DevPwrStateNotSet \" id= \"2 \"/>\" \\
 250: "<VariableRef name= \"Time \" />\" \\
 251: "<VariableRef name= \"SubS \" />\" \\
 252: "<VariableRef name= \"DevPwrState \" />\" \\
 253: "<VariableRef name= \"DevPwrStateSet \" />\" \\
 254: "</FaultMsg>\" \\
 255: "</Command>\" \\
 256: "<Notification>\" \\
 257: "<DataMsg name= \"DevPwrHK \" id= \"3 \" msgArrival= \"PERIODIC \"/>\" \\
 258: "<Qualifier name= \"telemetryLevel \" value= \"1 \"/>\" \\
 259: "<VariableRef name= \"Time \" />\" \\
 260: "<VariableRef name= \"SubS \" />\" \\
 261: "<VariableRef name= \"DevPwrState \" />\" \\
 262: "<VariableRef name= \"DevPwrStateSet \" />\" \\
 263: "</DataMsg>\" \\
 264: "</Notification>\" \\
 265: "<Request>\" \

```

266: "<CommandMsg name= \"getPowerInMode \" id= \"4 \" />\" \
267: "<DataReplyMsg name= \"powerInMode \" id= \"5 \">\" \
268: "<VariableRef name= \"modePowers \"/>\" \
269: "</DataReplyMsg>\" \
270: "</Request>\" \
271: "</Interface>\" \
272: "" \
273: "<Interface name= \"CmpSOH \" id= \"6 \">\" \
274: "<Qualifier name= \"TemperatureLoKeepout \" value= \"-20.0 \" units= \"degC \"/>\" \
275: "<Qualifier name= \"TemperatureLoWarning \" value= \"-10.0 \" units= \"degC \"/>\" \
276: "<Qualifier name= \"TemperatureHiWarning \" value= \"50.0 \" units= \"degC \"/>\" \
277: "<Qualifier name= \"TemperatureHiKeepout \" value= \"60.0 \" units= \"degC \"/>\" \
278: "<Variable kind= \"Time \" name= \"Time \" format= \"UINT32 \" units= \"Seconds \" />\" \
279: "<Variable kind= \"SubSeconds \" name= \"SubS \" units= \"Counts \" format= \"UINT32 \"
scaleFactor= \".0001 \" scaleUnits= \"Seconds \" />\" \
280: "<Variable name= \"DeviceTemperature \" kind= \"temperature \" format= \"FLOAT32 \" units=
\"degC \" />\" \
281: "<Request>\" \
282: "<CommandMsg name= \"GetDeviceTemperature \" id= \"1 \" />\" \
283: "<DataReplyMsg name= \"DeviceTempReply \" id= \"2 \">\" \
284: "<VariableRef name= \"Time \" />\" \
285: "<VariableRef name= \"SubS \" />\" \
286: "<VariableRef name= \"DeviceTemperature \"/>\" \
287: "</DataReplyMsg>\" \
288: "</Request>\" \
289: "<Notification>\" \
290: "<DataMsg name= \"DeviceTemp \" id= \"3 \" msgArrival= \"PERIODIC \" msgRate= \"1 \">\" \
291: "<Qualifier name= \"telemetryLevel \" value= \"1 \" />\" \
292: "<VariableRef name= \"Time \" />\" \
293: "<VariableRef name= \"SubS \" />\" \
294: "<VariableRef name= \"DeviceTemperature \"/>\" \
295: "</DataMsg>\" \
296: "</Notification>\" \
297: "</Interface>\" \
298: "</xTEDS>\" \
299: ""
300:
301: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/DigitalSS.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                               xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="AdcoleNRLDigitalSunSensorxTeds"
4:   version="2.0">
5:   <Device name="AdcoleNRLDigitalSunSensor" kind="fss" description="A single 2-Axis Digital Sun
Sensor" />
6:
7:   <Interface name="DigitalSunSensorInterface" id="1">
8:     <Qualifier name="headID" value="0"/>
9:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
10:    <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
11:    <Variable name="Sun_Presence" kind="Valid" format="UINT08">
12:      <Drange name="Sun_PresenceEnum">
13:        <Option name="SunPresent" value="1" description="This sensor can see the sun" />
14:        <Option name="SunNotPresent" value="0" description="Sun not visible from this
sensor" />
15:      </Drange>
16:    </Variable>
17:    <Variable name="SunAngle" kind="SunAngle" units="deg" format="FLOAT32" length="2"
rangeMin="-128"
18:      rangeMax="128" accuracy="0.25">
19:      <Qualifier name="Frame_Measured" value="DVF" />
20:    </Variable>
21:    <Notification>
22:      <DataMsg id="1" name="AngleToSun" msgArrival="PERIODIC" msgRate="100">
23:        <Qualifier name="telemetryLevel" value="1"/>
24:        <VariableRef name="Time"/>
25:        <VariableRef name="SubS"/>
26:        <VariableRef name="Sun_Presence" />
27:        <VariableRef name="SunAngle" />
28:      </DataMsg>
29:    </Notification>
30:  </Interface>
31:
32:  <Interface name="DevPwr" id="2">
33:    <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
34:    <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
```

```

35: <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
36: <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
37: <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
38:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
39: <Variable name="DevPwrState" kind="Power_State" format="UINT08">
40:     <Drange name="DevPwrStateEnum">
41:         <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
42:         <Option name="DevPwrON" value="1" description="Device may draw full power." />
43:     </Drange>
44: </Variable>
45: <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
46:     <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
47:         <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
48:         <Option name="DevPwrON" value="1" description="Device may draw full power." />
49:     </Drange>
50: </Variable>
51: <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
52: <Command>
53:     <CommandMsg name="DevPwrSetState" id="1">
54:         <VariableRef name="DevPwrStateSet" />
55:     </CommandMsg>
56:     <FaultMsg name="DevPwrStateNotSet" id="2">
57:         <VariableRef name="Time" />
58:         <VariableRef name="SubS" />
59:         <VariableRef name="DevPwrState" />
60:         <VariableRef name="DevPwrStateSet" />
61:     </FaultMsg>
62: </Command>
63: <Notification>
64:     <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
65:         <Qualifier name="telemetryLevel" value="1"/>
66:         <VariableRef name="Time" />
67:         <VariableRef name="SubS" />
68:         <VariableRef name="DevPwrState" />
69:         <VariableRef name="DevPwrStateSet" />
70:     </DataMsg>
71: </Notification>
72: <Request>
73:     <CommandMsg name="getPowerInMode" id="4" />

```

```

74:   <DataReplyMsg name="powerInMode" id="5">
75:     <VariableRef name="modePowers"/>
76:   </DataReplyMsg>
77: </Request>
78: </Interface>
79:
80: <Interface name="CmpSOH" id="3">
81:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
82:   <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
83:   <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
84:   <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
85:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
86:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
87:   <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
88:   <Request>
89:     <CommandMsg name="GetDeviceTemperature" id="1" />
90:     <DataReplyMsg name="DeviceTempReply" id="2">
91:       <VariableRef name="Time" />
92:       <VariableRef name="SubS" />
93:       <VariableRef name="DeviceTemperature"/>
94:     </DataReplyMsg>
95:   </Request>
96:   <Notification>
97:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
98:       <Qualifier name="telemetryLevel" value="1"/>
99:       <VariableRef name="Time" />
100:      <VariableRef name="SubS" />
101:      <VariableRef name="DeviceTemperature"/>
102:    </DataMsg>
103:  </Notification>
104: </Interface>
105: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/BIT.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:         xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd"
name="BITxTEDS" description="Built-In Test interface templates" version="0.1">
4:
5:     <Application name="Example" kind="REFERENCE_ONLY" description="A null application. The
purpose of this xTEDS is to define templates for BIT interfaces."/>
6:
7:     <Interface name="FunctionalTest" id="1" description="The Functional Built-In Test is a basic
functionality test. Typically initiated manually">
8:
9:         <Variable name="functionalTestResultCode" kind="STATUS" format="UINT08" description="The
result of running a Built-In Functional Test">
10:             <Drange name="functionalTestResultCodes" description="An enumeration of all test results">
11:                 <Option value="0" name="NOT_EXECUTED" description="The test has not been executed."/>
12:                 <Option value="1" name="SUCCESS" description="The test ran successfully."/>
13:                 <Option value="2" name="FAILURE" description="The test failed."/>
14:                 <!-- Note: FAILURE is a placeholder for defining specific failures. Typically all diagnosable
failures are enumerated. -->
15:             </Drange>
16:         </Variable>
17:
18:         <Variable kind="Time" name="functionalTestSeconds" format="UINT32" units="Seconds"
description="Time test was last executed."/>
19:         <Variable kind="SubSeconds" name="functionalTestSubSeconds" units="Counts"
format="UINT32" scaleFactor=".0001" scaleUnits="Seconds"
20:         description="Time test was last executed."/>
21:
22:     <Request>
23:         <CommandMsg name="runFunctionalTest" id="001" description="Run the Functional BIT and
report results."/>
24:         <DataReplyMsg name="functionalTestResult" id="002">
25:             <VariableRef name="functionalTestResultCode"/>
26:         </DataReplyMsg>
27:     </Request>
28:
29:     <Request>
30:         <CommandMsg name="getLastFunctionalTestResult" id="003" description="Run the Functional
BIT and report results."/>
31:         <DataReplyMsg name="lastFunctionalTestResult" id="004">
```



```

32:     <VariableRef name="functionalTestSeconds"/>
33:     <VariableRef name="functionalTestSubSeconds"/>
34:     <VariableRef name="functionalTestResultCode"/>
35: </DataReplyMsg>
36: </Request>
37:
38: <Notification>
39:     <DataMsg name="functionalTestResultChange" id="005" msgArrival="EVENT"
description="Functional Test result has changed.">
40:     <VariableRef name="functionalTestSeconds"/>
41:     <VariableRef name="functionalTestSubSeconds"/>
42:     <VariableRef name="functionalTestResultCode"/>
43: </DataMsg>
44: </Notification>
45:
46: </Interface>
47:
48: <Interface id="2" name="PerformanceTest" description="The Performance Built-In Test is a more
extensive operational test. Typically initiated manually.">
49:
50:     <Variable name="performanceTestResultCode" kind="STATUS" format="UINT08"
description="The result of running a Built-In Performance Test">
51:     <Drange name="performanceTestResultCodes" description="An enumeration of all test results">
52:     <Option value="0" name="NOT_EXECUTED" description="The test has not been executed."/>
53:     <Option value="1" name="SUCCESS" description="The test ran successfully."/>
54:     <Option value="2" name="FAILURE" description="The test failed."/>
55:     <!-- Note: FAILURE is a placeholder for defining specific failures. Typically all diagnosable
failures are enumerated. -->
56:     </Drange>
57: </Variable>
58:
59:     <Variable kind="Time" name="performanceTestSeconds" format="UINT32" units="Seconds"
description="Time test was last executed."/>
60:     <Variable kind="SubSeconds" name="performanceTestSubSeconds" units="Counts"
format="UINT32" scaleFactor=".0001" scaleUnits="Seconds"
61:     description="Time test was last executed."/>
62:
63: <Request>
64:     <CommandMsg name="runPerformanceTest" id="001" description="Run the Performance BIT
and report results."/>
65:     <DataReplyMsg name="performanceTestResult" id="002">
66:     <VariableRef name="performanceTestResultCode"/>

```

```

67:     </DataReplyMsg>
68: </Request>
69:
70: <Request>
71:     <CommandMsg name="getLastPerformanceTestResult" id="003" description="Run the
Performance BIT and report results. ">
72:     <DataReplyMsg name="lastPerformanceTestResult" id="004">
73:     <VariableRef name="performanceTestSeconds"/>
74:     <VariableRef name="performanceTestSubSeconds"/>
75:     <VariableRef name="performanceTestResultCode"/>
76:     </DataReplyMsg>
77: </Request>
78:
79: <Notification>
80:     <DataMsg name="performanceTestResultChange" id="005" msgArrival="EVENT"
description="Performance Test result has changed.">
81:     <VariableRef name="performanceTestSeconds"/>
82:     <VariableRef name="performanceTestSubSeconds"/>
83:     <VariableRef name="performanceTestResultCode"/>
84: </DataMsg>
85: </Notification>
86:
87: </Interface>
88:
89: <Interface id="3" name="OnOrbitCheckout" description="On-Orbit Checkout verifies operational
readiness. Autonomously initiated by OOCE.">
90:
91:     <Variable name="onOrbitCheckoutResultCode" kind="STATUS" format="UINT08"
description="The result of running an On-Orbit Checkout">
92:     <Drange name="onOrbitCheckoutResultCodes" description="An enumeration of all test results">
93:     <Option value="0" name="NOT_EXECUTED" description="The On-Orbit Checkout has not
been executed."/>
94:     <Option value="1" name="SUCCESS" description="The On-Orbit Checkout ran
successfully."/>
95:     <Option value="2" name="FAILURE" description="The On-Orbit Checkout failed."/>
96:     <!-- Note: FAILURE is a placeholder for defining specific failures. Typically all diagnosable
failures are enumerated. -->
97:     </Drange>
98: </Variable>
99:
100: <Variable kind="Time" name="onOrbitCheckoutSeconds" format="UINT32" units="Seconds"
description="TimeOn-Orbit Checkout was last executed."/>

```

```

101:         <Variable kind="SubSeconds" name="onOrbitCheckoutSubSeconds" units="Counts"
format="UINT32" scaleFactor=".0001" scaleUnits="Seconds"
102:         description="Time On-Orbit Checkout was last executed."/>
103:
104:     <Request>
105:         <CommandMsg name="runOnOrbitCheckout" id="001" description="Run the On-Orbit
Checkout and report results."/>
106:         <DataReplyMsg name="onOrbitCheckoutResult" id="002">
107:             <VariableRef name="onOrbitCheckoutResultCode"/>
108:         </DataReplyMsg>
109:     </Request>
110:
111:     <Request>
112:         <CommandMsg name="getLastOnOrbitCheckoutResult" id="003" description="Run the On-
Orbit Checkout and report results."/>
113:         <DataReplyMsg name="lastOnOrbitCheckoutResult" id="004">
114:             <VariableRef name="onOrbitCheckoutSeconds"/>
115:             <VariableRef name="onOrbitCheckoutSubSeconds"/>
116:             <VariableRef name="onOrbitCheckoutResultCode"/>
117:         </DataReplyMsg>
118:     </Request>
119:
120:     <Notification>
121:         <DataMsg name="onOrbitCheckoutResultChange" id="005" msgArrival="EVENT"
description="On-Orbit Checkout result has changed.">
122:             <VariableRef name="onOrbitCheckoutSeconds"/>
123:             <VariableRef name="onOrbitCheckoutSubSeconds"/>
124:             <VariableRef name="onOrbitCheckoutResultCode"/>
125:         </DataMsg>
126:     </Notification>
127:
128: </Interface>
129:
130: </xTEDS>

```

File: sdm/app/test/DMTests/xTEDSRegTests/GPS.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:           <xTEDS                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS    ../Schema/xTEDS02.xsd"
name="BasicGPSxTEDS"
4:   version="2.0">
5:   <Device name="BasicGPSReceiver" kind="sgr" description="A GPS receiver that produces
pseudorange measurements only" />
6:   <Interface name="GPSBasic" id="1">
7:     <Qualifier name="headID" value="0"/>
8:     <Variable name="numVisibleSats" kind="listSize" format="INT16" description="Number of
GPS satellites visible in this observation" />
9:     <Variable name="recvTime" kind="time" format="INT32" units="s" description="Receiver
clock time since GPS epoch">
10:       <Qualifier name="timeFrame" value="GPST1" />
11:     </Variable>
12:     <Variable name="PRNs" kind="ID" format="INT16" length="28" description="list of PRN
numbers of the visible GPS satellites">
13:       <Qualifier name="representation" value="array" />
14:     </Variable>
15:     <Variable name="PRN" kind="ID" format="INT16" description="PRN of a single GPS sat
generating a NAV message" />
16:     <Variable name="Prange" kind="distance" format="FLOAT32" length="28" units="km"
description="Pseudorange measurements for each visible GPS satellite">
17:       <Qualifier name="representation" value="array" />
18:     </Variable>
19:     <Variable name="epochTime" kind="epoch" format="INT16" length="6"
description="YMDHMS definition of the current GPS epoch time">
20:       <Qualifier name="representation" value="array" />
21:     </Variable>
22:     <Variable name="nav" kind="navMessage" format="FLOAT32" length="28" description="Nav
message from this satellite: ephemeris, clock corrections">
23:       <Qualifier name="representation" value="array" />
24:     </Variable>
25:     <Notification>
26:       <DataMsg name="GPSSatPrange" msgArrival="PERIODIC" msgRate="10" id="1">
27:         <Qualifier name="telemetryLevel" value="1"/>
28:         <VariableRef name="numVisibleSats" />
29:         <VariableRef name="recvTime" />
30:         <VariableRef name="PRNs" />
31:         <VariableRef name="Prange" />
```

```

32:     </DataMsg>
33: </Notification>
34: <Notification>
35:     <DataMsg name="GPSNavMessage" msgArrival="PERIODIC" msgRate="1" id="2">
36:         <VariableRef name="PRN" />
37:         <VariableRef name="epochTime" />
38:         <VariableRef name="nav" />
39:     </DataMsg>
40: </Notification>
41: </Interface>
42: <Interface name="DevPwr" id="2">
43:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
44:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
45:     <Qualifier name="CurrentHiWarning" value="3.5" units="A"/>
46:     <Qualifier name="CurrentHiKeepout" value="3.5" units="A"/>
47:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
48:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
49:     <Variable name="DevPwrState" kind="Power_State" format="UINT08">
50:         <Drange name="DevPwrStateEnum">
51:             <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
52:             <Option name="DevPwrON" value="1" description="Device may draw full power." />
53:         </Drange>
54:     </Variable>
55:     <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" id="2">
56:         <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
57:             <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
58:             <Option name="DevPwrON" value="1" description="Device may draw full power." />
59:         </Drange>
60:     </Variable>
61:     <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
62:     <Command>
63:         <CommandMsg name="DevPwrSetState" id="1">
64:             <VariableRef name="DevPwrStateSet" />
65:         </CommandMsg>
66:         <FaultMsg name="DevPwrStateNotSet" id="2">
67:             <VariableRef name="Time" />
68:             <VariableRef name="SubS" />
69:             <VariableRef name="DevPwrState" />
70:             <VariableRef name="DevPwrStateSet" />

```

```

71:    </FaultMsg>
72: </Command>
73: <Notification>
74:   <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
75:     <Qualifier name="telemetryLevel" value="1"/>
76:     <VariableRef name="Time" />
77:     <VariableRef name="SubS" />
78:     <VariableRef name="DevPwrState" />
79:     <VariableRef name="DevPwrStateSet" />
80:   </DataMsg>
81: </Notification>
82: <Request>
83:   <CommandMsg name="getPowerInMode" id="4" />
84:   <DataReplyMsg name="powerInMode" id="5">
85:     <VariableRef name="modePowers"/>
86:   </DataReplyMsg>
87: </Request>
88: </Interface>
89:
90: <Interface name="CmpSOH" id="3">
91:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
92:   <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
93:   <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
94:   <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
95:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
96:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
97:   <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
98:   <Request>
99:     <CommandMsg name="GetDeviceTemperature" id="1" />
100:    <DataReplyMsg name="DeviceTempReply" id="2">
101:      <VariableRef name="Time" />
102:      <VariableRef name="SubS" />
103:      <VariableRef name="DeviceTemperature"/>
104:    </DataReplyMsg>
105:   </Request>
106:   <Notification>
107:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
108:       <Qualifier name="telemetryLevel" value="1"/>
109:       <VariableRef name="Time" />
110:       <VariableRef name="SubS" />

```

111: <VariableRef name="DeviceTemperature"/>
112: </DataMsg>
113: </Notification>
114: </Interface>
115: </xTEDS>

File: sdm/app/test/DMTests/xTEDSRegTests/ChargeBatteries.h

```
1: #ifndef _CHARGE_BATTERIES_XTEDS_H
2: #define _CHARGE_BATTERIES_XTEDS_H
3:
4: #define _STRING_CHARGE_BATTERIES_XTEDS \
5: "<?xml version= \"1.0\" encoding= \"UTF-8\"?>\" \
6:   "<xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
   \"http://www.w3.org/2001/XMLSchema-instance \"\" \
7:   \"xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS  xTEDS02.xsd \"  name=
   \"ActivityAgentXTEDS \" description= \"ActivityAgent xTEDS \" version= \"2.3\">\" \
8:   \"\" \
9:   "<Application name= \"ChargeBatteries \" kind= \"AutonomyFlightSoftware \" description=
   \"Autonomous Tasking Executive (ATE), ActivityAgent \"/>\" \
10:   \"\" \
11:   "<!-- Note: An ActivityAgent must implement the ActivityInterface and should implement the
   ActivityAgentStatusInterface. -->\" \
12:   \"\" \
13:   "<Interface name= \"ActivityInterface \" id= \"1 \"><!-- This is a template for an ActivityAgent
   Interface. Unique activity parameters must be added as necessary. -->\" \
14:   \"\" \
15:   "<Variable name= \"ActivityName \" kind= \"tbd \" format= \"UINT08 \" length= \"33 \"><!-- 33-
   byte, null terminated string -->\" \
16:   "<Variable name= \"ActivityId \" kind= \"ID \" format= \"UINT32 \">\" \
17:   "<Variable name= \"ActivityStatus \" kind= \"Status \" format= \"INT16 \">\" \
18:   "<Drange name= \"ActivityStatusEnum \">\" \
19:   "<Option value= \"0 \" name= \"SCHEDULE_FAILURE \"><!-- The activity could not be scheduled
   as requested -->\" \
20:   "<Option value= \"1 \" name= \"WAITING \"><!-- The activity has been inserted into the schedule --
   >\" \
21:   "<Option value= \"2 \" name= \"ENABLED \"><!-- The activity has been sent a command to execute
   -->\" \
22:   "<Option value= \"3 \" name= \"TERMINATED \"><!-- The activity has been sent a command to
   abort -->\" \
23:   "<Option value= \"4 \" name= \"EXECUTING \"><!-- The activity is currently executing -->\" \
24:   "<Option value= \"5 \" name= \"DONE_FAILURE \"><!-- The activity has completed abnormally --
   >\" \
25:   "<Option value= \"6 \" name= \"DONE_SUCCESS \"><!-- The activity has completed normally -->\"
   \
26:   "<Option value= \"7 \" name= \"DONE_NOT_EXECUTED \"><!-- The activity has terminated
   without executing -->\" \
27:   "</Drange>\" \
28:   "</Variable>\" \
29:   \"\" \
```



```

30: "<!-- Add Variables for the unique activity parameters -->" \
31: "" \
32: "<Command> <!-- Provides the capability to request an Activity from the ground -->" \
33: "<CommandMsg name= \"RequestActivityCmd \" id= \"001 \" description= \"Determine activity state variables and request to be scheduled \">" \
34: "<VariableRef name= \"ActivityId \"/><!-- Use 0 for on-board requests. The ActivityId will then be assigned by the ActivityManager -->" \
35: "" \
36: "<!-- Add VariableRefs for the unique activity parameters -->" \
37: "" \
38: "</CommandMsg>" \
39: "</Command>" \
40: "" \
41: "<Request>" \
42: "<CommandMsg name= \"RequestActivity \" id= \"002 \" description= \"Determine activity state variables and request to be scheduled \">" \
43: "<VariableRef name= \"ActivityId \"/><!-- Use 0 for on-board requests. The ActivityId will then be assigned by the ActivityManager -->" \
44: "" \
45: "<!-- Add VariableRefs for the unique activity parameters -->" \
46: "" \
47: "</CommandMsg>" \
48: "<DataReplyMsg name= \"RequestActivityReply \" id= \"003 \">" \
49: "<VariableRef name= \"ActivityId \"/>" \
50: "<VariableRef name= \"ActivityStatus \"/>" \
51: "</DataReplyMsg>" \
52: "</Request>" \
53: "" \
54: "<Command> <!-- Provides the capability to update an Activity from the ground -->" \
55: "<CommandMsg name= \"UpdateRequestCmd \" id= \"004 \" description= \"Update activity state variables and request a schedule update if necessary \">" \
56: "<VariableRef name= \"ActivityId \"/>" \
57: "" \
58: "<!-- Add VariableRefs for the unique activity parameters -->" \
59: "" \
60: "</CommandMsg>" \
61: "</Command>" \
62: "" \
63: "<Request>" \
64: "<CommandMsg name= \"UpdateRequest \" id= \"005 \" description= \"Update activity state variables and request a schedule update if necessary \">" \
65: "<VariableRef name= \"ActivityId \"/>" \

```

```

66: "" \
67: "<!-- Add VariableRefs for the unique activity parameters -->" \
68: "" \
69: "</CommandMsg>" \
70: "<DataReplyMsg name= \"UpdateRequestReply \" id= \"006 \">" \
71: "<VariableRef name= \"ActivityId \"/>" \
72: "<VariableRef name= \"ActivityStatus \"/>" \
73: "</DataReplyMsg>" \
74: "</Request>" \
75: "" \
76: "<Command>" \
77: "<CommandMsg name= \"Reschedule \" id= \"007 \" description= \"The activity has been removed
from the schedule and needs to be rescheduled \">" \
78: "<VariableRef name= \"ActivityId \"/>" \
79: "</CommandMsg>" \
80: "</Command>" \
81: "" \
82: "<Command>" \
83: "<CommandMsg name= \"Delete \" id= \"008 \" description= \"Delete the activity \">" \
84: "<VariableRef name= \"ActivityId \"/>" \
85: "</CommandMsg>" \
86: "</Command>" \
87: "" \
88: "<Command>" \
89: "<CommandMsg name= \"Execute \" id= \"009 \" description= \"Execute the activity \">" \
90: "<VariableRef name= \"ActivityId \"/>" \
91: "</CommandMsg>" \
92: "</Command>" \
93: "" \
94: "<Command>" \
95: "<CommandMsg name= \"Abort \" id= \"010 \" description= \"Abort execution of the activity \">" \
96: "<VariableRef name= \"ActivityId \"/>" \
97: "</CommandMsg>" \
98: "</Command>" \
99: "" \
100: "<Command>" \
101: "<CommandMsg name= \"SendActivityStatusMsg \" id= \"011 \" description= \"Send the
ActivityStatusMsg DataMsg \">" \
102: "<VariableRef name= \"ActivityId \"/>" \
103: "</CommandMsg>" \
104: "</Command>" \

```

```

105: "" \
106: "<Notification>" \
107: "<DataMsg name= \"ActivityStatusMsg\" id= \"012\" msgArrival= \"EVENT \">>" \
108: "<Qualifier value= \"1\" name= \"telemetryLevel \"/>" \
109: "<VariableRef name= \"ActivityId \"/>" \
110: "<VariableRef name= \"ActivityName \"/>" \
111: "<VariableRef name= \"ActivityStatus \"/>" \
112: "</DataMsg>" \
113: "</Notification>" \
114: "" \
115: "</Interface>" \
116: "" \
117: "<Interface name= \"ActivityAgentStatusInterface\" id= \"2 \">>" \
118: "" \
119: "<Variable name= \"ActivityAgentStatus\" kind= \"Status\" format= \"INT16 \">>" \
120: "<Drange name= \"ActivityAgentStatusEnum \">>" \
121: "<Option value= \"0\" name= \"NOT_INITIALIZED \"/><!-- The ActivityAgent has not been
successfully initialized -->" \
122: "<Option value= \"1\" name= \"INITIALIZING \"/><!-- The ActivityAgent is in the process of
initializing -->" \
123: "<Option value= \"2\" name= \"RUNNING \"/><!-- The ActivityAgent is initialized and is running -
->" \
124: "<Option value= \"3\" name= \"TERMINATING \"/><!-- The ActivityAgent is shutting down -->"
\
125: "</Drange>" \
126: "</Variable>" \
127: "" \
128: "<Variable name= \"ActivityAgentName\" kind= \"tbd\" format= \"UINT08\" length= \"33 \"/><!--
33-byte, null terminated string -->" \
129: "" \
130: "<Variable name= \"ActivitiesCurrentlyScheduled\" kind= \"tbd\" format= \"UINT16 \"/>" \
131: "" \
132: "<Variable name= \"ActivitiesExecuted\" kind= \"tbd\" format= \"UINT16 \"/>" \
133: "<Variable name= \"ActivitiesExecutedSuccess\" kind= \"tbd\" format= \"UINT16 \"/>" \
134: "<Variable name= \"ActivitiesExecutedFailure\" kind= \"tbd\" format= \"UINT16 \"/>" \
135: "<Variable name= \"ActivitiesDeleted\" kind= \"tbd\" format= \"UINT16 \"/>" \
136: "" \
137: "<Variable name= \"RequestActivityReceived\" kind= \"tbd\" format= \"UINT16 \"/>" \
138: "<Variable name= \"RequestActivityAccepted\" kind= \"tbd\" format= \"UINT16 \"/>" \
139: "<Variable name= \"RequestActivitySuccess\" kind= \"tbd\" format= \"UINT16 \"/>" \
140: "<Variable name= \"RequestActivityFailure\" kind= \"tbd\" format= \"UINT16 \"/>" \
141: "" \

```

142: "<Variable name= \"UpdateRequestReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 143: "<Variable name= \"UpdateRequestAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 144: "<Variable name= \"UpdateRequestSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 145: "<Variable name= \"UpdateRequestFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 146: \"\" \\
 147: "<Variable name= \"ExecuteReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 148: "<Variable name= \"ExecuteAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 149: "<Variable name= \"ExecuteSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 150: "<Variable name= \"ExecuteFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 151: \"\" \\
 152: "<Variable name= \"RescheduleReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 153: "<Variable name= \"RescheduleAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 154: "<Variable name= \"RescheduleSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 155: "<Variable name= \"RescheduleFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 156: \"\" \\
 157: "<Variable name= \"DeleteReceived \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 158: "<Variable name= \"DeleteAccepted \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 159: "<Variable name= \"DeleteSuccess \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 160: "<Variable name= \"DeleteFailure \" kind= \"tbd \" format= \"UINT16 \"/>\" \\
 161: \"\" \\
 162: "<Command>\" \\
 163: "<CommandMsg name= \"SendActivityAgentStatusMsg \" id= \"001 \" description= \"Send the ActivityAgentStatusMsg DataMsg \"/>\" \\
 164: "</Command>\" \\
 165: \"\" \\
 166: "<Notification>\" \\
 167: "<DataMsg name= \"ActivityAgentStatusMsg \" id= \"002 \" msgArrival= \"EVENT \"/>\" \\
 168: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>\" \\
 169: "<VariableRef name= \"ActivityAgentStatus \"/>\" \\
 170: "<VariableRef name= \"ActivityAgentName \"/>\" \\
 171: "<VariableRef name= \"ActivitiesCurrentlyScheduled \"/>\" \\
 172: "<VariableRef name= \"ActivitiesExecuted \"/>\" \\
 173: "<VariableRef name= \"ActivitiesExecutedSuccess \"/>\" \\
 174: "<VariableRef name= \"ActivitiesExecutedFailure \"/>\" \\
 175: "<VariableRef name= \"ActivitiesDeleted \"/>\" \\
 176: "<VariableRef name= \"RequestActivityReceived \"/>\" \\
 177: "<VariableRef name= \"RequestActivityAccepted \"/>\" \\
 178: "<VariableRef name= \"RequestActivitySuccess \"/>\" \\
 179: "<VariableRef name= \"RequestActivityFailure \"/>\" \\
 180: "<VariableRef name= \"UpdateRequestReceived \"/>\" \\
 181: "<VariableRef name= \"UpdateRequestAccepted \"/>\" \

```

182: "<VariableRef name= \"UpdateRequestSuccess \"/>" \
183: "<VariableRef name= \"UpdateRequestFailure \"/>" \
184: "<VariableRef name= \"ExecuteReceived \"/>" \
185: "<VariableRef name= \"ExecuteAccepted \"/>" \
186: "<VariableRef name= \"ExecuteSuccess \"/>" \
187: "<VariableRef name= \"ExecuteFailure \"/>" \
188: "<VariableRef name= \"RescheduleReceived \"/>" \
189: "<VariableRef name= \"RescheduleAccepted \"/>" \
190: "<VariableRef name= \"RescheduleSuccess \"/>" \
191: "<VariableRef name= \"RescheduleFailure \"/>" \
192: "<VariableRef name= \"DeleteReceived \"/>" \
193: "<VariableRef name= \"DeleteAccepted \"/>" \
194: "<VariableRef name= \"DeleteSuccess \"/>" \
195: "<VariableRef name= \"DeleteFailure \"/>" \
196: "</DataMsg>" \
197: "</Notification>" \
198: "" \
199: "</Interface>" \
200: "" \
201: "<Interface name= \"ATEDebugInterface \" id= \"3 \">" \
202: "" \
203: "<!--" \
204: "Note: DebugLevel is a bit field with the following assigned values:" \
205: "DEBUG_ENTRY_AND_EXIT = 0x01," \
206: "DEBUG_ENTRY_PARAMETERS = 0x02," \
207: "DEBUG_EXIT_PARAMETERS = 0x04," \
208: "DEBUG_LEVEL_LOW = 0x08," \
209: "DEBUG_LEVEL_MEDIUM = 0x10," \
210: "DEBUG_LEVEL_HIGH = 0x20." \
211: "The values are OR'd to determine the debug information to be logged." \
212: "-->" \
213: "" \
214: "<Variable name= \"DebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
215: "<Variable name= \"CurrentDebugLevel \" kind= \"tbd \" format= \"UINT16 \"/>" \
216: "" \
217: "<Variable name= \"SetDebugLevelReceived \" kind= \"tbd \" format= \"UINT16 \"/>" \
218: "<Variable name= \"SetDebugLevelAccepted \" kind= \"tbd \" format= \"UINT16 \"/>" \
219: "<Variable name= \"SetDebugLevelSuccess \" kind= \"tbd \" format= \"UINT16 \"/>" \
220: "<Variable name= \"SetDebugLevelFailure \" kind= \"tbd \" format= \"UINT16 \"/>" \
221: "" \
222: "<Command>" \

```

```

223: "<CommandMsg name= \"SetDebugLevel \" id= \"001 \" description= \"Set the debug log verbosity
level \">" \
224: "<VariableRef name= \"DebugLevel \"/>" \
225: "</CommandMsg>" \
226: "</Command>" \
227: "" \
228: "<Command>" \
229: "<CommandMsg name= \"SendDebugStatus \" id= \"002 \"/>" \
230: "</Command>" \
231: "" \
232: "<Notification>" \
233: "<DataMsg name= \"DebugStatus \" id= \"003 \" msgArrival= \"EVENT \">" \
234: "<Qualifier value= \"1 \" name= \"telemetryLevel \"/>" \
235: "<VariableRef name= \"CurrentDebugLevel \"/>" \
236: "<VariableRef name= \"SetDebugLevelReceived \"/>" \
237: "<VariableRef name= \"SetDebugLevelAccepted \"/>" \
238: "<VariableRef name= \"SetDebugLevelSuccess \"/>" \
239: "<VariableRef name= \"SetDebugLevelFailure \"/>" \
240: "</DataMsg>" \
241: "</Notification>" \
242: "" \
243: "</Interface>" \
244: "" \
245: "<Interface name= \"TaskControlInterface \" id= \"4 \">" \
246: "" \
247: "<Command>" \
248: "<CommandMsg name= \"DestroyTask \" id= \"001 \"/>" \
249: "</Command>" \
250: "" \
251: "<!-- Other possible commands include Suspend, Resume, SetPriority, and SetHeartBeatPeriod -->"
\
252: "<!-- Other possible requests include GetPriority, and GetHeartBeatCount -->" \
253: "" \
254: "</Interface>" \
255: "" \
256: "</xTEDS>" \
257: ""
258:
259: #endif

```

File: sdm/app/test/DMTests/xTEDSRegTests/CSSAssy.xml

```
1: <?xml version="1.0" encoding="utf-8" ?>
2:         <xTEDS                                xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd"
name="DNet_CoarseSSAssy_xTEDS"
4:   version="2.1">
5:   <Device name="Coarse_sun_sensor_assembly" kind="cssa" description="An assembly of 4 coarse
sun sensors" />
6:
7:   <Interface id="1" name="CSSAssemblyInterface" description="Aggregate-level interface for the
assembly" >
8:     <Qualifier name="headID" value="0"/>
9:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
10:        <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
11:     <Variable name="SolutionAvailable" kind="boolean" format="UINT08">
12:       <Drange name="SolutionEnum" >
13:         <Option name="Yes" value="1"/>
14:         <Option name="No" value="0"/>
15:       </Drange>
16:     </Variable>
17:     <Variable name="dataQuality" kind="DataQuality" format="UINT08">
18:       <Drange name="DataQualityEnum">
19:         <Option name="dataBad" value="0" description="data is garbage or NaN." />
20:         <Option name="dataPoor" value="1" description="data quality is poor." />
21:         <Option name="dataGood" value="2" description="data is good." />
22:       </Drange>
23:     </Variable>
24:     <Variable name="AngleToSun" kind="sunLOS" format="FLOAT32" length="2" />
25:     <Notification>
26:       <DataMsg id="1" name="SunSolution" msgArrival="PERIODIC" msgRate="10">
27:         <Qualifier name="telemetryLevel" value="1"/>
28:         <VariableRef name="Time"/>
29:         <VariableRef name="SubS"/>
30:         <VariableRef name="SolutionAvailable"/>
31:         <VariableRef name="dataQuality"/>
32:         <VariableRef name="AngleToSun"/>
33:       </DataMsg>
34:     </Notification>
35:   </Interface>
```

```

36:
37: <Interface id="2" name="CSSAssyHeadInterface" description="Messaging for a single CSS head" >
38:   <Qualifier name="headID" value="1"/>
39:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
40:       <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
41:   <Variable kind="ID" name="HeadID" format="UINT08" />
42:   <Variable kind="boresightLOS" name="HeadLOS" format="FLOAT32" length="3" />
43:   <Variable kind="sunCOS" name="SunCos" format="FLOAT32" />
44:   <Variable name="SunPresence" kind="Valid" format="UINT08">
45:     <Drange name="Sun_PresenceEnum">
46:       <Option name="SunPresent" value="1" description="This sensor can see the sun" />
47:       <Option name="SunNotPresent" value="0" description="Sun not visible from this sensor" />
48:     </Drange>
49:   </Variable>
50:   <Request>
51:     <CommandMsg id="1" name="getHeadLOS" description="Returns the orientation of this head in
the sensor frame" />
52:     <DataReplyMsg id="2" name="HeadLOSReply">
53:       <VariableRef name="HeadID"/>
54:       <VariableRef name="HeadLOS"/>
55:     </DataReplyMsg>
56:   </Request>
57:   <Notification>
58:     <DataMsg id="3" name="sunConeAngle" msgArrival="PERIODIC" msgRate="10"
description="Observation of the sun from this sensor head">
59:       <Qualifier name="telemetryLevel" value="1"/>
60:       <VariableRef name="Time"/>
61:       <VariableRef name="SubS"/>
62:       <VariableRef name="SunPresence"/>
63:       <VariableRef name="SunCos"/>
64:     </DataMsg>
65:   </Notification>
66: </Interface>
67:
68: <Interface id="3" name="CSSAssyHeadInterface" description="Messaging for a single CSS head" >
69:   <Qualifier name="headID" value="2"/>
70:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
71:       <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
72:   <Variable kind="ID" name="HeadID" format="UINT08" />
73:   <Variable kind="boresightLOS" name="HeadLOS" format="FLOAT32" length="3" />

```



```

74: <Variable kind="sunCOS" name="SunCos" format="FLOAT32" />
75: <Variable name="SunPresence" kind="Valid" format="UINT08">
76:   <Drange name="Sun_PresenceEnum">
77:     <Option name="SunPresent" value="1" description="This sensor can see the sun" />
78:     <Option name="SunNotPresent" value="0" description="Sun not visible from this sensor" />
79:   </Drange>
80: </Variable>
81: <Request>
82:   <CommandMsg id="1" name="getHeadLOS" description="Returns the orientation of this head in
the sensor frame" />
83:   <DataReplyMsg id="2" name="HeadLOSReply">
84:     <VariableRef name="HeadID"/>
85:     <VariableRef name="HeadLOS"/>
86:   </DataReplyMsg>
87: </Request>
88: <Notification>
89:   <DataMsg id="3" name="sunConeAngle" msgArrival="PERIODIC" msgRate="10"
description="Observation of the sun from this sensor head">
90:     <Qualifier name="telemetryLevel" value="1"/>
91:     <VariableRef name="Time"/>
92:     <VariableRef name="SubS"/>
93:     <VariableRef name="SunPresence"/>
94:     <VariableRef name="SunCos"/>
95:   </DataMsg>
96: </Notification>
97: </Interface>
98:
99: <Interface id="4" name="CSSAssyHeadInterface" description="Messaging for a single CSS head" >
100:   <Qualifier name="headID" value="3"/>
101:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
102:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
103:   <Variable kind="ID" name="HeadID" format="UINT08" />
104:   <Variable kind="boresightLOS" name="HeadLOS" format="FLOAT32" length="3" />
105:   <Variable kind="sunCOS" name="SunCos" format="FLOAT32" />
106:   <Variable name="SunPresence" kind="Valid" format="UINT08">
107:     <Drange name="Sun_PresenceEnum">
108:       <Option name="SunPresent" value="1" description="This sensor can see the sun" />
109:       <Option name="SunNotPresent" value="0" description="Sun not visible from this sensor" />
110:     </Drange>
111:   </Variable>
112: <Request>

```

```

113:   <CommandMsg id="1" name="getHeadLOS" description="Returns the orientation of this head in
the sensor frame" />
114:   <DataReplyMsg id="2" name="HeadLOSReply">
115:     <VariableRef name="HeadID"/>
116:     <VariableRef name="HeadLOS"/>
117:   </DataReplyMsg>
118: </Request>
119: <Notification>
120:     <DataMsg id="3" name="sunConeAngle" msgArrival="PERIODIC" msgRate="10"
description="Observation of the sun from this sensor head">
121:       <Qualifier name="telemetryLevel" value="1"/>
122:       <VariableRef name="Time"/>
123:       <VariableRef name="SubS"/>
124:       <VariableRef name="SunPresence"/>
125:       <VariableRef name="SunCos"/>
126:     </DataMsg>
127: </Notification>
128: </Interface>
129:
130: <Interface id="5" name="CSSAssyHeadInterface" description="Messaging for a single CSS head" >
131:   <Qualifier name="headID" value="4"/>
132:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
133:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
134:   <Variable kind="ID" name="HeadID" format="UINT08" />
135:   <Variable kind="boresightLOS" name="HeadLOS" format="FLOAT32" length="3" />
136:   <Variable kind="sunCOS" name="SunCos" format="FLOAT32" />
137:   <Variable name="SunPresence" kind="Valid" format="UINT08">
138:     <Drange name="Sun_PresenceEnum">
139:       <Option name="SunPresent" value="1" description="This sensor can see the sun" />
140:       <Option name="SunNotPresent" value="0" description="Sun not visible from this sensor" />
141:     </Drange>
142:   </Variable>
143:   <Request>
144:     <CommandMsg id="1" name="getHeadLOS" description="Returns the orientation of this head
in the sensor frame" />
145:     <DataReplyMsg id="2" name="HeadLOSReply">
146:       <VariableRef name="HeadID"/>
147:       <VariableRef name="HeadLOS"/>
148:     </DataReplyMsg>
149:   </Request>
150:   <Notification>

```

```

151:         <DataMsg id="3" name="sunConeAngle" msgArrival="PERIODIC" msgRate="10"
description="Observation of the sun from this sensor head">
152:     <Qualifier name="telemetryLevel" value="1"/>
153:     <VariableRef name="Time"/>
154:     <VariableRef name="SubS"/>
155:     <VariableRef name="SunPresence"/>
156:     <VariableRef name="SunCos"/>
157: </DataMsg>
158: </Notification>
159: </Interface>
160:
161: <Interface name="DevPwr" id="6">
162:     <Qualifier name="CurrentLoKeepout" value="0.0" units="A"/>
163:     <Qualifier name="CurrentLoWarning" value="0.0" units="A"/>
164:     <Qualifier name="CurrentHiWarning" value="0.2" units="A"/>
165:     <Qualifier name="CurrentHiKeepout" value="0.3" units="A"/>
166:     <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
167:     <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
168:     <Variable name="DevPwrState" kind="Power_State" format="UINT08" >
169:         <Drange name="DevPwrStateEnum">
170:             <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
171:             <Option name="DevPwrON" value="1" description="Device may draw full power." />
172:         </Drange>
173:     </Variable>
174:     <Variable name="DevPwrStateSet" kind="Power_State" format="UINT08" >
175:         <Drange name="DevPwrStateEnumReference" description="This should be a reference to
DevPwrStateEnumeration.">
176:             <Option name="DevPwrOFF" value="0" description="All power to device is turned off." />
177:             <Option name="DevPwrON" value="1" description="Device may draw full power." />
178:         </Drange>
179:     </Variable>
180:     <Variable name="modePowers" kind="power" format="FLOAT32" units="W" length="2" />
181:     <Command>
182:         <CommandMsg name="DevPwrSetState" id="1">
183:             <VariableRef name="DevPwrStateSet" />
184:         </CommandMsg>
185:         <FaultMsg name="DevPwrStateNotSet" id="2">
186:             <VariableRef name="Time" />
187:             <VariableRef name="SubS" />
188:             <VariableRef name="DevPwrState" />
189:             <VariableRef name="DevPwrStateSet" />

```

```

190:    </FaultMsg>
191: </Command>
192: <Notification>
193:   <DataMsg name="DevPwrHK" id="3" msgArrival="PERIODIC">
194:     <Qualifier name="telemetryLevel" value="1"/>
195:     <VariableRef name="Time" />
196:     <VariableRef name="SubS" />
197:     <VariableRef name="DevPwrState" />
198:     <VariableRef name="DevPwrStateSet" />
199:   </DataMsg>
200: </Notification>
201: <Request>
202:   <CommandMsg name="getPowerInMode" id="4" />
203:   <DataReplyMsg name="powerInMode" id="5">
204:     <VariableRef name="modePowers"/>
205:   </DataReplyMsg>
206: </Request>
207: </Interface>
208:
209: <Interface name="CmpSafety" id="7">
210:   <Qualifier name="TemperatureLoKeepout" value="-20.0" units="degC"/>
211:   <Qualifier name="TemperatureLoWarning" value="-10.0" units="degC"/>
212:   <Qualifier name="TemperatureHiWarning" value="50.0" units="degC"/>
213:   <Qualifier name="TemperatureHiKeepout" value="60.0" units="degC"/>
214:   <Variable kind="Time" name="Time" format="UINT32" units="Seconds" />
215:   <Variable kind="SubSeconds" name="SubS" units="Counts" format="UINT32"
scaleFactor=".0001" scaleUnits="Seconds" />
216:   <Variable name="DeviceTemperature" kind="temperature" format="FLOAT32" units="degC" />
217:   <Request>
218:     <CommandMsg name="GetDeviceTemperature" id="1" />
219:     <DataReplyMsg name="DeviceTempReply" id="2">
220:       <VariableRef name="Time" />
221:       <VariableRef name="SubS" />
222:       <VariableRef name="DeviceTemperature"/>
223:     </DataReplyMsg>
224:   </Request>
225:   <Notification>
226:     <DataMsg name="DeviceTemp" id="3" msgArrival="PERIODIC" msgRate="1">
227:       <Qualifier name="telemetryLevel" value="1"/>
228:       <VariableRef name="Time" />
229:       <VariableRef name="SubS" />

```

230: <VariableRef name="DeviceTemperature"/>
231: </DataMsg>
232: </Notification>
233: </Interface>
234:
235: </xTEDS>

File: sdm/app/test/DMTests/xTEDSRegTests/IDS.xml

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xTEDS name="IDSxTEDS"
3:   xmlns="http://www.interfacecontrol.com/SPA/xTEDS"
4:   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5:   xsi:schemaLocation="http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd"
6:   version="2.0">
7:
8:   <Application name="IntelligentDataStore" kind="Software">
9:     <Qualifier name="author" value="Ken_Center"/>
10:    <Qualifier name="company" value="Design_Net_Engineering"/>
11:  </Application>
12:
13:  <Interface name="FileAccessInterface" id="1">
14:    <Variable name="FileHandle" format="UINT16" kind="Handle" description="This field is used
to identify file requests."/>
15:    <Variable name="PathName" format="INT08" kind="PathName_String" length="80"
description="This field is the null-terminated path to the file."/>
16:    <Variable name="FileName" format="INT08" kind="FileName_String" length="80"
description="This field is the null-terminated filename of the request."/>
17:    <Variable name="FileBuffer" format="UINT08" kind="File_Section" length="8047"
description="This field is the buffer containing the portion of the file requested."/>
18:    <Variable name="StatusCode" format="UINT08" kind="Status_Code" description="This field is
a status corresponding to a fault.">
19:      <Drange name="StatusCodeTypes">
20:        <Option name="OperationOK" value="1"/>
21:        <Option name="InvalidHandle" value="2"/>
22:        <Option name="FileNotAvailable" value="3"/>
23:        <Option name="InvalidOffset" value="4"/>
24:        <Option name="CouldNotObtainHandle" value="5"/>
25:      </Drange>
26:    </Variable>
27:    <Variable name="FileFlag" format="UINT08" kind="Flag" description="This field represents
the type of handle to obtain.">
28:      <Drange name="FlagTypes">
29:        <Option name="ReadOnly" value="1"/>
30:        <Option name="WriteOnly" value="2"/>
31:        <Option name="ReadWrite" value="3"/>
32:      </Drange>
33:    </Variable>
```

```

34:     <Variable name="ByteOffset" format="UINT32" kind="Offset" description="This field is the
byte offset for reading and writing to files."/>
35:     <Variable name="Length" format="UINT32" kind="Length" description="This field is the
number of bytes to read/write from the offset."/>
36:     <Request>
37:         <CommandMsg name="OpenFileHandle" id="1">
38:             <VariableRef name="PathName"/>
39:             <VariableRef name="FileName"/>
40:             <VariableRef name="FileFlag"/>
41:         </CommandMsg>
42:         <DataReplyMsg name="OpenHandleReply" id="2">
43:             <VariableRef name="PathName"/>
44:             <VariableRef name="FileName"/>
45:             <VariableRef name="StatusCode"/>
46:             <VariableRef name="FileHandle"/>
47:         </DataReplyMsg>
48:     </Request>
49:     <Command>
50:         <CommandMsg name="CloseFileHandle" id="3">
51:             <VariableRef name="FileHandle"/>
52:         </CommandMsg>
53:         <FaultMsg name="CloseFileHandleError" id="4">
54:             <VariableRef name="FileHandle"/>
55:             <VariableRef name="StatusCode"/>
56:         </FaultMsg>
57:     </Command>
58:     <Request>
59:         <CommandMsg name="ReadPortion" id="5">
60:             <VariableRef name="FileHandle"/>
61:             <VariableRef name="ByteOffset"/>
62:             <VariableRef name="Length"/>
63:         </CommandMsg>
64:         <DataReplyMsg name="ReadReply" id="6">
65:             <VariableRef name="FileHandle"/>
66:             <VariableRef name="StatusCode"/>
67:             <VariableRef name="Length"/>
68:             <VariableRef name="FileBuffer"/>
69:         </DataReplyMsg>
70:     </Request>
71:     <Request>
72:         <CommandMsg name="WritePortion" id="7">

```

```

73:         <VariableRef name="FileHandle"/>
74:         <VariableRef name="ByteOffset"/>
75:         <VariableRef name="Length"/>
76:         <VariableRef name="FileBuffer"/>
77:     </CommandMsg>
78:     <DataReplyMsg name="WriteReply" id="8">
79:         <VariableRef name="FileHandle"/>
80:         <VariableRef name="StatusCode"/>
81:         <VariableRef name="Length"/>
82:     </DataReplyMsg>
83: </Request>
84: </Interface>
85:
86: <Interface name="FileManagementInterface" id="2">
87:     <Variable name="FileListTextBuffer" format="UINT08" length="8047" kind="Text_Buffer"
description="Text buffer containing File Listing" />
88:     <Variable name="FileName" format="INT08" length="80" kind="FileName_String"
description="Name of File" />
89:     <Variable name="PathName" format="INT08" length="80" kind="PathName_String"
description="File Path Text" />
90:     <Variable name="FromFileName" format="INT08" length="80" kind="FileName_String"
description="Name of Originating File" />
91:     <Variable name="FromPathName" format="INT08" length="80" kind="PathName_String"
description="Originating File Path Text" />
92:     <Variable name="ToFileName" format="INT08" length="80" kind="FileName_String"
description="Name of Destination File" />
93:     <Variable name="ToPathName" format="INT08" length="80" kind="PathName_String"
description="Destination File Path Text" />
94:     <Variable name="FileSize" format="UINT32" kind="File_Size_Bytes" description="Size of File
in Bytes" />
95:     <Variable name="FileOwner" format="UINT08" length="16" kind="File_Owner"
description="File owner text string" />
96:     <Variable name="FileLocked" format="UINT08" kind="File_Lock_State"
description="Indicates whether file is in use - 1 is available, 2 is locked" />
97:     <Variable name="FilePermissions" format="UINT08" length="16" kind="File_Permissions"
description="File permissions text string" />
98:     <Variable name="StatusCode" format="UINT08" kind="Status_Code" description="Fail code
for file management actions">
99:         <Drange name="StatusCodeTypes">
100:             <Option name="OperationOK" value="1"/>
101:             <Option name="InvalidPath" value="2"/>
102:             <Option name="InvalidFile" value="3"/>
103:             <Option name="PathExists" value="4"/>

```



```

104:         <Option name="FileExists" value="5"/>
105:         <Option name="InvalidPathAndFile" value="6"/>
106:         <Option name="InsufficientMemory" value="7"/>
107:         <Option name="PathNotEmpty" value="8"/>
108:     </Drange>
109: </Variable>
110: <Variable      name="FileListCode"      format="UINT08"      kind="File_List_Code"
description="Type of File Listing - brief or complete">
111:     <Drange name="FileListCodeTypes">
112:         <Option name="Brief" value="1"/>
113:         <Option name="Complete" value="2"/>
114:     </Drange>
115: </Variable>
116:
117: <Request>
118:     <CommandMsg name="FileList" id="1">
119:         <VariableRef name="PathName"/>
120:         <VariableRef name="FileListCode"/>
121:     </CommandMsg>
122:     <DataReplyMsg name="FileListReply" id="2">
123:         <VariableRef name="PathName"/>
124:         <VariableRef name="StatusCode"/>
125:         <VariableRef name="FileListTextBuffer"/>
126:     </DataReplyMsg>
127: </Request>
128: <Request>
129:     <CommandMsg name="FileInfo" id="3">
130:         <VariableRef name="PathName"/>
131:         <VariableRef name="FileName"/>
132:     </CommandMsg>
133:     <DataReplyMsg name="FileInfoReply" id="4">
134:         <VariableRef name="PathName"/>
135:         <VariableRef name="FileName"/>
136:         <VariableRef name="StatusCode"/>
137:         <VariableRef name="FileSize"/>
138:         <VariableRef name="FileOwner"/>
139:         <VariableRef name="FileLocked"/>
140:         <VariableRef name="FilePermissions"/>
141:     </DataReplyMsg>
142: </Request>
143: <Request>

```

```

144:         <CommandMsg name="FileCopy" id="5">
145:             <VariableRef name="FromPathName"/>
146:             <VariableRef name="FromFileName"/>
147:             <VariableRef name="ToPathName"/>
148:             <VariableRef name="ToFileName"/>
149:         </CommandMsg>
150:         <DataReplyMsg name="FileCopyReply" id="6">
151:             <VariableRef name="FromPathName"/>
152:             <VariableRef name="FromFileName"/>
153:             <VariableRef name="ToPathName"/>
154:             <VariableRef name="ToFileName"/>
155:             <VariableRef name="StatusCode"/>
156:         </DataReplyMsg>
157:     </Request>
158:     <Request>
159:         <CommandMsg name="FileMove" id="7">
160:             <VariableRef name="FromPathName"/>
161:             <VariableRef name="FromFileName"/>
162:             <VariableRef name="ToPathName"/>
163:             <VariableRef name="ToFileName"/>
164:         </CommandMsg>
165:         <DataReplyMsg name="FileMoveReply" id="8">
166:             <VariableRef name="FromPathName"/>
167:             <VariableRef name="FromFileName"/>
168:             <VariableRef name="ToPathName"/>
169:             <VariableRef name="ToFileName"/>
170:             <VariableRef name="StatusCode"/>
171:         </DataReplyMsg>
172:     </Request>
173:     <Request>
174:         <CommandMsg name="FileDelete" id="9">
175:             <VariableRef name="PathName"/>
176:             <VariableRef name="FileName"/>
177:         </CommandMsg>
178:         <DataReplyMsg name="FileDeleteReply" id="10">
179:             <VariableRef name="PathName"/>
180:             <VariableRef name="FileName"/>
181:             <VariableRef name="StatusCode"/>
182:         </DataReplyMsg>
183:     </Request>
184:     <Request>

```

```

185:         <CommandMsg name="PathCreate" id="11">
186:             <VariableRef name="PathName"/>
187:         </CommandMsg>
188:         <DataReplyMsg name="PathCreateReply" id="12">
189:             <VariableRef name="PathName"/>
190:             <VariableRef name="StatusCode"/>
191:         </DataReplyMsg>
192:     </Request>
193:     <Request>
194:         <CommandMsg name="PathDelete" id="13">
195:             <VariableRef name="PathName"/>
196:         </CommandMsg>
197:         <DataReplyMsg name="PathDeleteReply" id="14">
198:             <VariableRef name="PathName"/>
199:             <VariableRef name="StatusCode"/>
200:         </DataReplyMsg>
201:     </Request>
202: </Interface>
203:
204: <Interface name="BufferInterface" id="3">
205:     <Variable name="BufferName" format="INT08" length="80" kind="Buffer_Name_String"
description="Requested name of buffer" />
206:     <Variable name="ElementLength" format="UINT16" kind="Data_Size_Bytes"
description="Size of data buffer in Bytes" />
207:     <Variable name="MaxNumElements" format="UINT16" kind="Max_Number_Elements"
description="Maximum number of elements that log can contain" />
208:     <Variable name="RcvSensorID" format="UINT32" kind="Sensor_ID" description="Sensor
ID associated with receiving buffer" />
209:     <Variable name="RcvIpAddr" format="UINT32" kind="IP_Address" description="IP
Address associated with receiving buffer" />
210:     <Variable name="RcvPortID" format="UINT16" kind="Port_ID" description="Port ID
associated with receiving buffer" />
211:     <Variable name="MySensorID" format="UINT32" kind="Sensor_ID" description="Sensor
ID of application consuming drain data" />
212:     <Variable name="MyIpAddr" format="UINT32" kind="IP_Address" description="IP
Address of application consuming drain data" />
213:     <Variable name="MyPortID" format="UINT16" kind="Port_ID" description="Port ID of
application consuming drain data" />
214:     <Variable name="BufferFillLevel" format="FLOAT32" kind="Fill_Level"
description="Buffer fill level in percent" />
215:     <Variable name="NumTotalElements" format="UINT32" kind="Num_Elements"
description="Number of elements stored in buffer" />

```

216: <Variable name="NumReadElements" format="UINT32" kind="Num_Read_Elements" description="Number of elements that have been read but not cleared" />

217: <Variable name="NumUnreadElements" format="UINT32" kind="Num_Unread_Elements" description="Number of elements that have not been read" />

218: <Variable name="SourceSensorID" format="UINT32" kind="Sensor_ID" description="Sensor ID of qualified search data" />

219: <Variable name="SourceIpAddr" format="UINT32" kind="IP_Address" description="IP Address of qualified search data" />

220: <Variable name="SourcePortID" format="UINT16" kind="Port_ID" description="Port ID of qualified search data" />

221: <Variable name="DataSource" format="UINT32" kind="Component_ID" description="ComponentID of message origination for buffer searches, -1 indicates no constraint" />

222: <Variable name="MinTime" format="INT32" kind="Time" units="Seconds" description="Min time range value for buffer searches, -1 indicates no constraint" />

223: <Variable name="MaxTime" format="INT32" kind="Time" units="Seconds" description="Max time range value for buffer searches, -1 indicates no constraint" />

224: <Variable name="SourceSearchFlag" format="UINT08" kind="Source_Search_Flag" description="Flag indicating search source data, 1 is all, 2 uses Source data from SourceSensorID/SourceIpAddress/SourcePortID variables" >

225: <Drange name="SearchFlagCodes">

226: <Option name="AllSources" value="1"/>

227: <Option name="SelectSource" value="2"/>

228: </Drange>

229: </Variable>

230: <Variable name="ReadMask" format="UINT08" kind="Read_Mask_Mode" description="Defines whether previously read elements are to be saerched" >

231: <Drange name="ReadMaskCodes">

232: <Option name="UnreadOnly" value="1"/>

233: <Option name="ReadOnly" value="2"/>

234: <Option name="UnreadAndRead" value="3"/>

235: </Drange>

236: </Variable>

237: <Variable name="DrainStyle" format="UINT08" kind="Drain_Mode" description="Defines whether to drain the buffer Last-In-First-Out or First-In-First-Out" >

238: <Drange name="DrainStyleCodes">

239: <Option name="FIFO" value="1"/>

240: <Option name="LIFO" value="2"/>

241: </Drange>

242: </Variable>

243: <Variable name="BufferInterfaceResponseCode" format="UINT08" kind="Response_Code" description="Failure Response for Buffer Operations" >

244: <Drange name="BufferInterfaceResponseCodeTypes">

245: <Option name="OperationOK" value="1"/>

246: <Option name="NameInUse" value="2"/>

```

247:         <Option name="InsufficientMemory" value="3"/>
248:         <Option name="InvalidTimeRange" value="4"/>
249:         <Option name="NoSuchBuffer" value="5"/>
250:         <Option name="NoElementAvailable" value="6"/>
251:     </Drange>
252: </Variable>
253:
254: <Request>
255:     <CommandMsg name="BufferCreate" id="1">
256:         <VariableRef name="BufferName" />
257:         <VariableRef name="ElementLength" />
258:         <VariableRef name="MaxNumElements" />
259:     </CommandMsg>
260:     <DataReplyMsg name="BufferCreateReply" id="2">
261:         <VariableRef name="BufferName"/>
262:         <VariableRef name="BufferInterfaceResponseCode"/>
263:         <VariableRef name="RcvSensorID"/>
264:         <VariableRef name="RcvIpAddr"/>
265:         <VariableRef name="RcvPortID"/>
266:     </DataReplyMsg>
267: </Request>
268: <Request>
269:     <CommandMsg name="BufferSetDrainBehavior" id="3">
270:         <VariableRef name="BufferName" />
271:         <VariableRef name="DataSource" />
272:         <VariableRef name="MinTime" />
273:         <VariableRef name="MaxTime" />
274:         <VariableRef name="ReadMask" />
275:     </CommandMsg>
276:     <DataReplyMsg name="SetDrainFail" id="4">
277:         <VariableRef name="BufferName" />
278:         <VariableRef name="BufferInterfaceResponseCode"/>
279:     </DataReplyMsg>
280: </Request>
281: <Request>
282:     <CommandMsg name="BufferGetDrainBehavior" id="5">
283:         <VariableRef name="BufferName" />
284:     </CommandMsg>
285:     <DataReplyMsg name="GetDrainResponse" id="6">
286:         <VariableRef name="BufferName" />
287:         <VariableRef name="BufferInterfaceResponseCode"/>

```

```

288:         <VariableRef name="SourceSearchFlag" />
289:         <VariableRef name="SourceSensorID" />
290:         <VariableRef name="SourceIpAddr" />
291:         <VariableRef name="SourcePortID" />
292:         <VariableRef name="MinTime" />
293:         <VariableRef name="MaxTime" />
294:         <VariableRef name="ReadMask" />
295:     </DataReplyMsg>
296: </Request>
297: <Request>
298:     <CommandMsg name="BufferGetStatus" id="7">
299:         <VariableRef name="BufferName" />
300:     </CommandMsg>
301:     <DataReplyMsg name="GetBufferStatusResponse" id="8">
302:         <VariableRef name="BufferName" />
303:         <VariableRef name="BufferInterfaceResponseCode"/>
304:         <VariableRef name="BufferFillLevel" />
305:         <VariableRef name="NumTotalElements" />
306:         <VariableRef name="NumReadElements" />
307:         <VariableRef name="NumUnreadElements" />
308:     </DataReplyMsg>
309: </Request>
310: <Request>
311:     <CommandMsg name="BufferDelete" id="9">
312:         <VariableRef name="BufferName" />
313:     </CommandMsg>
314:     <DataReplyMsg name="BufferDeleteResponse" id="10">
315:         <VariableRef name="BufferName" />
316:         <VariableRef name="BufferInterfaceResponseCode"/>
317:     </DataReplyMsg>
318: </Request>
319: <Request>
320:     <CommandMsg name="BufferGetNextElement" id="11">
321:         <VariableRef name="BufferName" />
322:         <VariableRef name="MySensorID"/>
323:         <VariableRef name="MyIpAddr"/>
324:         <VariableRef name="MyPortID"/>
325:         <VariableRef name="DrainStyle" />
326:     </CommandMsg>
327:     <DataReplyMsg name="BufferGetNextElementReply" id="12">
328:         <VariableRef name="BufferName" />

```

```

329:         <VariableRef name="ElementLength" />
330:         <VariableRef name="BufferInterfaceResponseCode"/>
331:     </DataReplyMsg>
332: </Request>
333: </Interface>
334:
335: <Interface name="LoggerInterface" id="4">
336:     <Variable name="LogName" format="INT08" length="80" kind="Logfile_String"
description="Requested name of log" />
337:     <Variable name="MaxEntryLength" format="UINT08" kind="Entry_Size_Chars"
description="Max size of a log entry in characters" />
338:     <Variable name="MaxEntries" format="UINT32" kind="Max_Log_Entries"
description="Maximum number of entries" />
339:     <Variable name="LogFillLevel" format="FLOAT32" kind="Fill_Level"
description="Percent of log filled with entries" />
340:     <Variable name="LogTimeSeconds" format="UINT32" kind="Timestamp_Seconds"
description="Seconds portion of Log Entry Timestamp" />
341:     <Variable name="LogTimeSubseconds" format="UINT32" kind="Timestamp_Subseconds"
description="Subseconds portion of Log Entry Timestamp" />
342:     <Variable name="LogEntry" format="INT08" length="80" kind="String" description="Text
data to be saved to Log" />
343:     <Variable name="LogInterfaceResponseCode" format="UINT08" kind="Response_Code"
description="Response for Log Interface" >
344:         <Drange name="LogInterfaceResponseCodeTypes">
345:             <Option name="OperationOK" value="1"/>
346:             <Option name="NameInUse" value="2"/>
347:             <Option name="InsufficientMemory" value="3"/>
348:             <Option name="NoSuchLog" value="4"/>
349:             <Option name="LogFull" value="5"/>
350:             <Option name="EntryTooLong" value="6"/>
351:             <Option name="LogAlreadyOpen" value="7"/>
352:         </Drange>
353:     </Variable>
354:
355: <Request>
356:     <CommandMsg name="LogCreate" id="1">
357:         <VariableRef name="LogName" />
358:         <VariableRef name="MaxEntryLength" />
359:         <VariableRef name="MaxEntries" />
360:     </CommandMsg>
361:     <DataReplyMsg name="LogCreateReply" id="2">
362:         <VariableRef name="LogName"/>
363:         <VariableRef name="LogInterfaceResponseCode"/>

```

```

364:         </DataReplyMsg>
365:     </Request>
366: <Request>
367:     <CommandMsg name="LogStatus" id="3">
368:         <VariableRef name="LogName" />
369:     </CommandMsg>
370:     <DataReplyMsg name="LogStatusReply" id="4">
371:         <VariableRef name="LogName"/>
372:         <VariableRef name="LogFillLevel"/>
373:         <VariableRef name="LogInterfaceResponseCode"/>
374:     </DataReplyMsg>
375: </Request>
376: <Request>
377:     <CommandMsg name="LogWrite" id="5">
378:         <VariableRef name="LogName" />
379:         <VariableRef name="LogTimeSeconds" />
380:         <VariableRef name="LogTimeSubseconds" />
381:         <VariableRef name="LogEntry" />
382:     </CommandMsg>
383:     <DataReplyMsg name="LogWriteReply" id="6">
384:         <VariableRef name="LogName"/>
385:         <VariableRef name="LogInterfaceResponseCode"/>
386:     </DataReplyMsg>
387: </Request>
388: <Request>
389:     <CommandMsg name="LogOpen" id="7">
390:         <VariableRef name="LogName" />
391:     </CommandMsg>
392:     <DataReplyMsg name="LogOpenReply" id="8">
393:         <VariableRef name="LogName"/>
394:         <VariableRef name="LogInterfaceResponseCode"/>
395:     </DataReplyMsg>
396: </Request>
397: <Request>
398:     <CommandMsg name="LogClose" id="9">
399:         <VariableRef name="LogName" />
400:     </CommandMsg>
401:     <DataReplyMsg name="LogCloseReply" id="10">
402:         <VariableRef name="LogName"/>
403:         <VariableRef name="LogInterfaceResponseCode"/>
404:     </DataReplyMsg>

```



```
405:     </Request>
406:     <Request>
407:         <CommandMsg name="LogDelete" id="11">
408:             <VariableRef name="LogName" />
409:         </CommandMsg>
410:         <DataReplyMsg name="LogDeleteReply" id="12">
411:             <VariableRef name="LogName"/>
412:             <VariableRef name="LogInterfaceResponseCode"/>
413:         </DataReplyMsg>
414:     </Request>
415:
416: </Interface>
417:
418: </xTEDS>
```

Listing from directory: sdm/asim_test

File: sdm/asim_test/asim_test.cpp

```
1: #include "../common/asim/ASIM.h"
2: #include "../common/message_defs.h"
3:
4: #include <stdio.h>
5: #include <unistd.h>
6: #include <pthread.h>
7: #include <ctype.h>
8: #include <string.h>
9:
10: ASIM sensor;
11:
12: void menu();
13: void init();
14: void reset();
15: void selftest();
16: void data();
17: void stream();
18: void poweron();
19: void powerdown();
20: void version();
21: void asim_command();
22: void xteds();
23: void cancel();
24: void timeattone();
25:
26: void* Listener(void*);
27:
28: int main(int argc, char** argv)
29: {
30:     pthread_t listener_thread;
31:     printf("ASIM Tester 0.1 \n");
32:
33:     if (argc == 3)
34:     {
35:         printf(" Opening %s", argv[1]);
36:         if (strcmp(argv[2], "debug") == 0)
37:         {
```

```

38:         sensor.SetDebug(4);
39:         printf(" in DEBUG mode. \n");
40:     }
41:     else
42:         printf(" \n");
43:     if (!sensor.Open(argv[1]))
44:     {
45:         printf(" Error: Could not open device %s \n",argv[1]);
46:         _exit(0);
47:     }
48: }
49: else if (argc == 2)
50: {
51:     if (!sensor.Open(argv[1]))
52:     {
53:         printf(" Error: Could not open device %s \n",argv[1]);
54:         _exit(0);
55:     }
56:     printf(" Opening %s \n",argv[1]);
57: }
58: else
59: {
60:     printf("Usage: %s device [debug] \n",argv[0]);
61:     _exit(0);
62: }
63: printf(" ASIM USB path: %s \n \n",sensor.USBLocation());
64: pthread_create(&listener_thread,NULL,&Listener,NULL);
65: usleep(50000);
66: init();
67: usleep(50000);
68: version();
69:
70: while(1)
71:     menu();
72: pthread_join(listener_thread,NULL);
73: }
74:
75: void* Listener(void*)
76: {
77:     char buf[BUFSIZE];
78:     unsigned short length;

```

```

79: int i;
80:
81: while(1)
82: {
83:     switch(sensor.Read(length,(unsigned char*)buf,sizeof(buf)))
84:     {
85:         case ASIM_STATUS:
86:             if (buf[0]&0x80) //first bit is set for an error
87:             {
88:                 printf("ASIM status ERROR: 0x%hhx \n",buf[0]);
89:                 if(buf[0]&0x40) //illegal command bit
90:                 {
91:                     printf("Illegal or unrecognized command \n");
92:                 }
93:                 if(buf[0]&0x20) //self test failure bit
94:                 {
95:                     printf("Self Test failed \n");
96:                 }
97:             }
98:             else
99:             {
100:                 printf("ASIM status OK: 0x%hhx \n",buf[0]);
101:             }
102:             if(buf[0]&0x10) //mode bit
103:             {
104:                 printf("ASIM mode: operational \n");
105:             }
106:             else
107:             {
108:                 printf("ASIM mode: idle \n");
109:             }
110:             break;
111:         case ASIM_XTEDS:
112:             printf(" \nASIM xTEDS: \n");
113:             for(i=0;i<length;++i)
114:             {
115:                 if(isprint(buf[i])||isspace(buf[i]))
116:                     printf("%c",buf[i]);
117:                 else
118:                     printf(" \nERROR: contains non-printable character %x \n",buf[i]);
119:             }

```

```

120:         printf(" \n");
121:         break;
122:     case ASIM_DATA:
123:         printf(" \nASIM data message (%hu bytes): (%hhd, %hhd) \n",length,buf[0],buf[1]);
124:         for(i=0;i<length-2;++i)
125:         {
126:             printf("%hhx ",buf[2+i]);
127:         }
128:         printf(" \n");
129:         break;
130:     case ASIM_VERSION:
131:         printf(" \nASIM version %hhd \n",buf[0]);
132:         break;
133:     case ASIM_ERROR:
134:         printf(" \nASIM read error \n");
135:         break;
136:     default:
137:         printf(" \nUnexpected ASIM message \n");
138:     }
139: }
140: }
141:
142: void menu(void)
143: {
144:     int command;
145:
146:     printf("1) Initialize \n");
147:     printf("2) Reset \n");
148:     printf("3) Self Test \n");
149:     printf("4) Data request \n");
150:     printf("5) Stream request \n");
151:     printf("6) Power On \n");
152:     printf("7) Power Down \n");
153:     printf("8) Version request \n");
154:     printf("9) Command \n");
155:     printf("10) xTEDS request \n");
156:     printf("11) Time at Tone \n");
157:     printf("12) Cancel stream \n");
158:     printf("13) Quit \n");
159:     printf("command: ");
160:     if(scanf("%d",&command)==0)

```

```
161:  {
162:      getchar();
163:  }
164:  switch(command)
165:  {
166:      case 1:
167:          init();
168:          break;
169:      case 2:
170:          reset();
171:          break;
172:      case 3:
173:          selftest();
174:          break;
175:      case 4:
176:          data();
177:          break;
178:      case 5:
179:          stream();
180:          break;
181:      case 6:
182:          poweron();
183:          break;
184:      case 7:
185:          powerdown();
186:          break;
187:      case 8:
188:          version();
189:          break;
190:      case 9:
191:          asim_command();
192:          break;
193:      case 10:
194:          xteds();
195:          break;
196:      case 11:
197:          timeattone();
198:          break;
199:      case 12:
200:          cancel();
201:          break;
```

```

202:     case 13:
203:         _exit(0);
204:     default:
205:         printf("Bad Command \n");
206:     }
207: }
208:
209: void init()
210: {
211:     printf("Initializing \n");
212:     if (!sensor.Initialize())
213:         printf("Error sending initialize message \n");
214: }
215:
216: void reset()
217: {
218:     printf("Reseting \n");
219:     if (!sensor.Reset())
220:         printf("Error sending reset message \n");
221: }
222:
223: void selftest()
224: {
225:     printf("Self Testing \n");
226:     if (!sensor.SelfTest())
227:         printf("Error sending self test message \n");
228: }
229:
230: void data()
231: {
232:     unsigned char msg_id;
233:     unsigned char interface_id;
234:     printf("Message id: ");
235:     scanf("%hhhd",&msg_id);
236:     printf("Interface id: ");
237:     scanf("%hhhd",&interface_id);
238:     printf("Requesting Data message id %hhhd in interface %hhhd \n",msg_id,interface_id);
239:     if (sensor.ReqData(interface_id,msg_id))
240:         printf("Error sending data request message \n");
241: }
242:

```

```

243: void stream()
244: {
245:     unsigned char msg_id;
246:     unsigned char interface_id;
247:     unsigned long count;
248:     printf("Message id: ");
249:     scanf("%hhhd",&msg_id);
250:     printf("Interface id: ");
251:     scanf("%hhhd",&interface_id);
252:     printf("Message count: ");
253:     scanf("%ld",&count);
254:     printf("Requesting Data message id %hhhd in interface %hhhd, %ld times
\n",msg_id,interface_id,count);
255:     if (!sensor.ReqStream(interface_id,msg_id,count))
256:         printf("Error sending stream request message \n");
257: }
258:
259: void poweron()
260: {
261:     printf("Power On \n");
262:     if (!sensor.PowerOn())
263:         printf("Error sending power on message \n");
264: }
265:
266: void powerdown()
267: {
268:     printf("Power Down \n");
269:     if (!sensor.PowerDown())
270:         printf("Error sending power down message \n");
271: }
272:
273: void version()
274: {
275:     printf("Version request \n");
276:     if (!sensor.ReqVersion())
277:         printf("Error sending version request message \n");
278: }
279:
280: void asim_command()
281: {
282:     printf("Command \n");

```



```

283:
284:  unsigned char msg_id;
285:  unsigned char interface_id;
286:  unsigned short length;
287:  unsigned char buf[64];
288:  printf("Interface id: ");
289:  scanf("%hhu",&interface_id);
290:  printf("Message id: ");
291:  scanf("%hhu",&msg_id);
292:  printf("Command length: ");
293:  scanf("%hu",&length);
294:  if (length > 59)
295:  {
296:      printf("Invalid length. \n");
297:      return;
298:  }
299:  for (unsigned short i = 0; i < length; i++)
300:  {
301:      printf("Byte %hu: ",i);
302:      scanf("%hhu",buf+i);
303:  }
304:  if (!sensor.Command(interface_id, msg_id, length, buf))
305:      printf("Error sending command message \n");
306: }
307:
308: void xteds()
309: {
310:     printf("xTEDS request \n");
311:     sensor.ReqxTEDS();
312: }
313:
314: void timeattone()
315: {
316:     printf("Time at Tone \n");
317:     long sec, usec;
318:
319:     printf("Seconds: ");
320:     scanf("%ld",&sec);
321:     printf("USeconds: ");
322:     scanf("%ld",&usec);
323:     printf("Time at tone: seconds %ld useconds %ld \n",sec,usec);

```

```

324:   if (!sensor.TimeAtTone(sec,usec))
325:       printf("Error sending time at tone message \n");
326: }
327:
328: void cancel()
329: {
330:     unsigned char msg_id;
331:     unsigned char interface_id;
332:     printf("Message id: ");
333:     scanf("%hhd",&msg_id);
334:     printf("Interface id: ");
335:     scanf("%hhd",&interface_id);
336:     printf("Cancel message id %hhd in interface %hhd \n",msg_id, interface_id);
337:     if (!sensor.Cancel(interface_id,msg_id))
338:         printf("Error sending cancel message \n");
339: }

```

File: sdm/asim_test/asim_test_gui.cpp

```
1: #include "../common/asim/ASIM.h"
2: #include "../common/message_defs.h"
3:
4: #include <stdio.h>
5: #include <unistd.h>
6: #include <pthread.h>
7: #include <ctype.h>
8: #include <curses.h>
9:
10: #include <string.h>
11: #include <stdlib.h>
12: #include <errno.h>
13:
14: #define MENU_WIDTH    23
15: #define MENU_HEIGHT   17
16:
17: ASIM sensor;
18:
19: bool RawMode;
20: int cur_line = 1;
21:
22: WINDOW* w_menu;
23: WINDOW* w_in;
24: WINDOW* w_out;
25:
26: void menu();
27: void init();
28: void reset();
29: void selftest();
30: void data();
31: void stream();
32: void poweron();
33: void powerdown();
34: void version();
35: void asim_command();
36: void xteds();
37: void cancel();
38: void timeattone();
39: void rawmodetoggle();
```

```

40:
41: void* Listener(void*);
42:
43: void drawGUI(void);
44: void drawMenu(void);
45: void drawInput(void);
46: void drawOutput(void);
47: void displayRead(void);
48: void displayRaw(void);
49:
50: int main(int argc,char** argv)
51: {
52: pthread_t listener_thread;
53: RawMode = false;
54: if (argc == 3)
55: {
56:     if(strcmp(argv[2],"debug")==0)
57:         sensor.SetDebug(4);
58: }
59: if (argc >= 2)
60: {
61:     fflush(NULL);
62:     if(!sensor.Open(argv[1]))
63:     {
64:         printf("Error: Could not open device %s \n",argv[1]);
65:         _exit(0);
66:     }
67:     fflush(NULL);
68:     if(!sensor.VerifyConnection())
69:     {
70:         printf("Error: Could not verify connection \n");
71:         _exit(0);
72:     }
73:
74: }
75: else
76: {
77:     printf ("ASIM Tester %s \n",SDM_VERSION);
78:     printf("Usage: %s device [debug] \n",argv[0]);
79:     _exit(0);
80: }

```

```

81:
82: /* initialize curses*/
83: initscr();
84: keypad(stdscr,TRUE);
85: nonl();
86: cbreak();
87: noecho();
88:
89: drawGUI();
90: pthread_create(&listener_thread,NULL,&Listener,NULL);
91: sleep(2);
92: //reset();
93: //sleep(1);
94: init();
95: sleep(1);
96: version();
97: while(1)
98:     menu();
99: pthread_join(listener_thread,NULL);
100:     return 0;
101: }
102:
103: void displayRaw(void)
104: {
105:     char buf[BUFSIZE];
106:     short length;
107:     int i;
108:
109:     int error_code;
110:     char error_str[80];
111:
112:     length = sensor.RawRead((unsigned char*)buf);
113:
114:
115:     if (cur_line >= LINES-4)
116:     {
117:         cur_line = 1;
118:         drawOutput();
119:     }
120:     error_code = errno;
121:     wmove(w_out,cur_line,1);

```

```

122:
123:  if(length < 0)
124:  {
125:      wprintw(w_out,"ASIM read error: %s",strerror_r(error_code,error_str,80));
126:  }
127:  else
128:  {
129:      wprintw(w_out,"%d bytes",length);
130:      if(isprint(buf[0]))
131:      {
132:          wprintw(w_out," (%c)",buf[0]);
133:      }
134:      wprintw(w_out,"%s",": \t");
135:      for(i=0;i<length;i++)
136:          wprintw(w_out,"%0.2hhx ",buf[i]);
137:  }
138:  cur_line++;
139:  wrefresh(w_out);
140: }
141:
142: void displayRead(void)
143: {
144:     char buf[BUFSIZE];
145:     unsigned short length;
146:     int i;
147:
148:     char msg_type;
149:
150:     int error_code;
151:     char error_str[80];
152:
153:     msg_type = sensor.Read(length,(unsigned char*)buf, sizeof(buf));
154:
155:     if (cur_line >= LINES-4)
156:     {
157:         cur_line = 1;
158:         drawOutput();
159:     }
160:
161:     error_code = errno;
162:     switch(msg_type)

```

```

163:  {
164:  case ASIM_STATUS:
165:      wmove(w_out,cur_line,1);
166:      if (buf[0]&0x80) //first bit is set for an error
167:      {
168:          wprintw(w_out,"ASIM status ERROR: %hhd",buf[0]);
169:          if(buf[0]&0x40) //illegal command bit
170:          {
171:              cur_line++;
172:              wmove(w_out,cur_line,1);
173:              wprintw(w_out,"Illegal or unrecognized command");
174:          }
175:          if(buf[0]&0x20) //self test failure bit
176:          {
177:              cur_line++;
178:              wmove(w_out,cur_line,1);
179:              wprintw(w_out,"Self Test failed");
180:          }
181:      }
182:      else
183:      {
184:          wprintw(w_out,"ASIM status OK: %hhd",buf[0]);
185:      }
186:      if(buf[0]&0x10) //mode bit
187:      {
188:          cur_line++;
189:          wmove(w_out,cur_line,1);
190:          wprintw(w_out,"ASIM mode: operational");
191:      }
192:      else
193:      {
194:          cur_line++;
195:          wmove(w_out,cur_line,1);
196:          wprintw(w_out,"ASIM mode: idle");
197:      }
198:      cur_line++;
199:      break;
200:  case ASIM_XTEDS:
201:      //clear output screen for the xTEDS
202:      cur_line = 1;
203:      drawOutput();

```

```

204: //display xTEDS
205: wmove(w_out,cur_line,1);
206: wprintw(w_out,"ASIM xTEDS (%d):",length);
207: cur_line++;
208: wmove(w_out,cur_line,1);
209: cur_line++;
210: for(i=0;i<length;++i)
211: {
212:     if(isprint(buf[i]))
213:     {
214:         wprintw(w_out,"%c",buf[i]);
215:     }
216:     else
217:     {
218:         if(iscntrl(buf[i]))
219:         {
220:             if(buf[i] == '\n')
221:             {
222:                 wmove(w_out,cur_line,1);
223:                 cur_line++;
224:             }
225:             if(buf[i] == '\t')
226:             {
227:                 wprintw(w_out," ");
228:             }
229:         }
230:         else
231:         {
232:             wmove(w_out,1,13);
233:             wprintw(w_out,"ERROR: contains non-printable character 0x%hhx",buf[i]);
234:             wmove(w_out,cur_line,1);
235:             cur_line++;
236:         }
237:     }
238: }
239: cur_line++;
240: break;
241: case ASIM_DATA:
242:     wmove(w_out,cur_line,1);
243:     wprintw(w_out,"ASIM data message (%d bytes): (%hhd,%hhd)",length,buf[0],buf[1]);
244:     cur_line++;

```



```

245:     wmove(w_out,cur_line,1);
246:     for(i=0;i<length-1;++i)
247:     {
248:         wprintw(w_out,"%hhx ",buf[1+i]);
249:     }
250:     cur_line++;
251:     break;
252: case ASIM_VERSION:
253:     wmove(w_out,cur_line,1);
254:     wprintw(w_out,"ASIM version %hhd",buf[0]);
255:     cur_line++;
256:     break;
257: case ASIM_ERROR:
258:     wmove(w_out,cur_line,1);
259:     wprintw(w_out,"ASIM read error: %s",strerror_r(error_code,error_str,80));
260:     cur_line++;
261:     break;
262: case ASIM_TIMEOUT:
263:     wmove(w_out,cur_line,1);
264:     wprintw(w_out,"ASIM read error: %s",strerror_r(error_code,error_str,80));
265:     cur_line++;
266:     break;
267: default:
268:     wmove(w_out,cur_line,1);
269:     wprintw(w_out,"Unexpected ASIM message");
270:     cur_line++;
271:     wmove(w_out,cur_line,1);
272:     wprintw(w_out,"%hhx %hd ",msg_type,length);
273:     for(i=0;i<length;++i)
274:     {
275:         wprintw(w_out,"%hhx ",buf[i]);
276:     }
277:     cur_line++;
278:     break;
279: }
280: wrefresh(w_out);
281:
282: }
283:
284: void* Listener(void*)
285: {

```

```

286: while(1)
287: {
288:     if(RawMode)
289:     {
290:         displayRaw();
291:     }
292:     else
293:     {
294:         displayRead();
295:     }
296: }
297: return NULL;
298: }
299:
300: void drawGUI(void)
301: {
302:     //make main border with title
303:     box(stdscr,0,0);
304:     move(0,1);
305:     wprintw(stdscr,"ASIM Tester %s",SDM_VERSION);
306:     w_menu = subwin(stdscr,MENU_HEIGHT,MENU_WIDTH,1,1);
307:     w_out = subwin(stdscr,LINES-2,COLS-2-MENU_WIDTH,1,MENU_WIDTH+1);
308:     w_in = subwin(stdscr,LINES-2-MENU_HEIGHT,MENU_WIDTH,MENU_HEIGHT+1,1);
309:     drawInput();
310:     drawOutput();
311:     drawMenu();
312:     wrefresh(stdscr);
313: }
314:
315: void drawInput(void)
316: {
317:     werase(w_in);
318:     box(w_in,0,0);
319:     wmove(w_in,0,1);
320:     waddstr(w_in,"ASIM Input");
321:     wrefresh(w_in);
322: }
323:
324: void drawOutput(void)
325: {
326:     werase(w_out);

```

```

327:   box(w_out,0,0);
328:   wmove(w_out,0,1);
329:   wprintw(w_out,"ASIM Output (USB path:%s) ",sensor.USBLocation());
330:   if(RawMode)
331:       wprintw(w_out,"%s","Raw Mode");
332:   wrefresh(w_out);
333: }
334:
335: void drawMenu(void)
336: {
337:     int ITEMS = 15;
338:     int i;
339:     char* menu[]={
340:         "(I)nitalize","(R)eset","(S)tream request",
341:         "(P)ower On","Power (D)own","(V)ersion request",
342:         "Co(m)mand","(x)TEDS request","Time at T(o)ne",
343:         "Self (T)est","D(a)ta request","(C)ancel stream"," ","Ra(w) Mode","(Q)uit"
344:     };
345:
346:     //make menu border with title
347:     box(w_menu,0,0);
348:     //wmove(w_menu,0,1);
349:     //waddstr(w_menu,"Menu");
350:     //make menu
351:     for(i=0;i<ITEMS;i++)
352:     {
353:         wmove(w_menu,i+1,1);
354:         waddstr(w_menu,menu[i]);
355:     }
356:     wrefresh(w_menu);
357: }
358:
359: void menu(void)
360: {
361:     char command;
362:     while(1)
363:     {
364:         command = getch();
365:
366:         switch(command)
367:         {

```

```
368:     case 'T':
369:     case 'i':
370:         init();
371:         break;
372:     case 'R':
373:     case 'r':
374:         reset();
375:         break;
376:     case 'T':
377:     case 't':
378:         selftest();
379:         break;
380:     case 'A':
381:     case 'a':
382:         data();
383:         break;
384:     case 'S':
385:     case 's':
386:         stream();
387:         break;
388:     case 'P':
389:     case 'p':
390:         poweron();
391:         break;
392:     case 'D':
393:     case 'd':
394:         powerdown();
395:         break;
396:     case 'V':
397:     case 'v':
398:         version();
399:         break;
400:     case 'M':
401:     case 'm':
402:         asim_command();
403:         break;
404:     case 'X':
405:     case 'x':
406:         xteds();
407:         break;
408:     case 'O':
```

```

409:     case 'o':
410:         timeattone();
411:         break;
412:     case 'C':
413:     case 'c':
414:         cancel();
415:         break;
416:     case 'Q':
417:     case 'q':
418:         endwin();
419:         _exit(0);
420:     case 'W':
421:     case 'w':
422:         rawmodetoggle();
423:         break;
424:     default:
425:         wprintw(w_out,"Bad Command \n");
426:     }
427: }
428: }
429:
430: void rawmodetoggle()
431: {
432:     RawMode = !RawMode;
433:     box(w_out,0,0);
434:     wmove(w_out,0,1);
435:     wprintw(w_out,"ASIM Output (USB path:%s) ",sensor.USBLocation());
436:     if(RawMode)
437:         wprintw(w_out,"%s","Raw Mode");
438:     wrefresh(w_out);
439:
440: }
441:
442: void init()
443: {
444:     drawInput();
445:     wmove(w_in,1,1);
446:     if(sensor.Initialize())
447:     {
448:         waddstr(w_in,"Initialize Sent");
449:     }

```

```

450:     else
451:     {
452:         waddstr(w_in,"Initialize Failed");
453:     }
454:     wrefresh(w_in);
455: }
456:
457: void reset()
458: {
459:     drawInput();
460:     wmove(w_in,1,1);
461:     if(sensor.Reset())
462:     {
463:         waddstr(w_in,"Reset Sent");
464:     }
465:     else
466:     {
467:         waddstr(w_in,"Reset Failed");
468:     }
469:     wrefresh(w_in);
470: }
471:
472: void selftest()
473: {
474:     drawInput();
475:     wmove(w_in,1,1);
476:     if(sensor.SelfTest())
477:     {
478:         waddstr(w_in,"Self Test Sent");
479:     }
480:     else
481:     {
482:         waddstr(w_in,"Self Test Failed");
483:     }
484:     wrefresh(w_in);
485: }
486:
487: void data()
488: {
489:     unsigned char msg_id;
490:     unsigned char interface_id;

```

```

491:
492:   drawInput();
493:   wmove(w_in,1,1);
494:   waddstr(w_in,"Interface id: ");
495:   echo();
496:   wscanw(w_in,"%hhd",&interface_id);
497:   noecho();
498:   wmove(w_in,2,1);
499:   waddstr(w_in,"Message id: ");
500:   echo();
501:   wscanw(w_in,"%hhd",&msg_id);
502:   noecho();
503:   wmove(w_in,3,1);
504:   if(sensor.ReqData(interface_id,msg_id))
505:   {
506:       wprintw(w_in,"Data Requested (%hhd)",msg_id);
507:   }
508:   else
509:   {
510:       waddstr(w_in,"Data Request Failed");
511:   }
512:   wrefresh(w_in);
513:
514: }
515:
516: void stream()
517: {
518:     unsigned char msg_id;
519:     unsigned long count;
520:     unsigned char interface_id;
521:
522:     drawInput();
523:     wmove(w_in,1,1);
524:     waddstr(w_in,"Interface id: ");
525:     echo();
526:     wscanw(w_in,"%hhd",&interface_id);
527:     noecho();
528:     wmove(w_in,2,1);
529:     waddstr(w_in,"Message id: ");
530:     echo();
531:     wscanw(w_in,"%hhd",&msg_id);

```

```

532:  noecho();
533:  wmove(w_in,3,1);
534:  waddstr(w_in,"Message count: ");
535:  echo();
536:  wscanw(w_in,"%ld",&count);
537:  noecho();
538:  wmove(w_in,4,1);
539:  if(sensor.ReqStream(interface_id,msg_id,count))
540:  {
541:      wprintw(w_in,"Data Requested (%hhd)",msg_id);
542:  }
543:  else
544:  {
545:      waddstr(w_in,"Stream Request Failed");
546:  }
547:  wrefresh(w_in);
548: }
549:
550: void poweron()
551: {
552:     drawInput();
553:     wmove(w_in,1,1);
554:     if(sensor.PowerOn())
555:     {
556:         waddstr(w_in,"Power On Sent");
557:     }
558:     else
559:     {
560:         waddstr(w_in,"Power On Failed");
561:     }
562:     wrefresh(w_in);
563: }
564:
565: void powerdown()
566: {
567:     drawInput();
568:     wmove(w_in,1,1);
569:     if(sensor.PowerDown())
570:     {
571:         waddstr(w_in,"Power Down Sent");
572:     }

```



```

573:     else
574:     {
575:         waddstr(w_in,"Power Down Failed");
576:     }
577:     wrefresh(w_in);
578: }
579:
580: void version()
581: {
582:     drawInput();
583:     wmove(w_in,1,1);
584:     if(sensor.ReqVersion())
585:     {
586:         waddstr(w_in,"Version Request Sent");
587:     }
588:     else
589:     {
590:         waddstr(w_in,"Version Request Failed");
591:     }
592:     wrefresh(w_in);
593: }
594:
595: void asim_command()
596: {
597:     unsigned char cmd_id;
598:     short length;
599:     unsigned char* data;
600:     int i;
601:     unsigned char interface_id;
602:
603:     drawInput();
604:     wmove(w_in,1,1);
605:     waddstr(w_in,"Interface id: ");
606:     echo();
607:     wscanw(w_in,"%hhd",&interface_id);
608:     noecho();
609:     wmove(w_in,2,1);
610:     waddstr(w_in,"Command id: ");
611:     echo();
612:     wscanw(w_in,"%hhd",&cmd_id);
613:     noecho();

```

```

614:  wmove(w_in,3,1);
615:  waddstr(w_in,"Command length: ");
616:  echo();
617:  wscanw(w_in,"%hd",&length);
618:  noecho();
619:  data = (unsigned char*)malloc(sizeof(unsigned char)*length);
620:  for(i=0;i<length;i++)
621:  {
622:      wmove(w_in,4+i,1);
623:      waddstr(w_in,"Command byte: ");
624:      echo();
625:      wscanw(w_in,"%hhd",&(data[i]));
626:      noecho();
627:  }
628:  wmove(w_in,5+i,1);
629:  if(sensor.Command(interface_id,cmd_id,length,data))
630:  {
631:      wprintw(w_in,"Command %hhd Sent",cmd_id);
632:  }
633:  else
634:  {
635:      wprintw(w_in,"Command %hhd Failed",cmd_id);
636:  }
637:  free(data);
638:  wrefresh(w_in);
639: }
640:
641: void xteds()
642: {
643:     drawInput();
644:     wmove(w_in,1,1);
645:     if(sensor.ReqxTEDS())
646:     {
647:         waddstr(w_in,"xTEDS Request Sent");
648:     }
649:     else
650:     {
651:         waddstr(w_in,"xTEDS Request Failed");
652:     }
653:     wrefresh(w_in);
654: }

```

```

655:
656: void timeattone()
657: {
658:     long sec,usec;
659:
660:     drawInput();
661:     wmove(w_in,1,1);
662:     waddstr(w_in,"seconds: ");
663:     echo();
664:     wscanw(w_in,"%ld",&sec);
665:     noecho();
666:     wmove(w_in,2,1);
667:     waddstr(w_in,"microsecons: ");
668:     echo();
669:     wscanw(w_in,"%ld",&usec);
670:     noecho();
671:     wmove(w_in,3,1);
672:     wprintw(w_in,"Time at Tone:");
673:     wmove(w_in,4,1);
674:     wprintw(w_in,"%ld s %ld us",sec,usec);
675:     wmove(w_in,5,1);
676:     if(sensor.TimeAtTone(sec,usec))
677:     {
678:         waddstr(w_in,"Time at Tone Sent");
679:     }
680:     else
681:     {
682:         waddstr(w_in,"Time at Tone Failed");
683:     }
684:     wrefresh(w_in);
685: }
686:
687: void cancel()
688: {
689:     unsigned char msg_id;
690:     unsigned char interface_id;
691:
692:     drawInput();
693:     wmove(w_in,1,1);
694:     waddstr(w_in,"Interface id: ");
695:     echo();

```

```

696:  wscanw(w_in,"%hhd",&interface_id);
697:  noecho();
698:  wmove(w_in,2,1);
699:  waddstr(w_in,"Message id: ");
700:  echo();
701:  wscanw(w_in,"%hhd",&msg_id);
702:  noecho();
703:  wmove(w_in,3,1);
704:  wprintw(w_in,"Data Canceled (%hhd)",msg_id);
705:  wmove(w_in,4,1);
706:  if(sensor.Cancel(interface_id,msg_id))
707:  {
708:      waddstr(w_in,"Cancel Sent");
709:  }
710:  else
711:  {
712:      waddstr(w_in,"Cancel Failed");
713:  }
714:  wrefresh(w_in);
715: }

```

File: sdm/asim_test/Makefile

```
1: # Makefile for asim test system
2:
3: include ../$(MAKEFILE_DEFS)
4:
5: .PHONY:    clean distclean
6:
7: all: asim_test_gui asim_test
8:
9: asim_test_gui: asim_test_gui.o
10: $(CXX) $(CXXFLAGS) -o $@ $^ $(BOOSTFLAGS) -lpthread -lncurses -L../common -lSDM
11:
12: asim_test:  asim_test.o
13: $(CXX) $(CXXFLAGS) -o $@ $^ -lpthread -L../common -lSDM -lboost_regex
14:
15: asim_test_gui.o: asim_test_gui.cpp
16: $(CXX) $(CXXFLAGS) -c $<
17:
18: asim_test.o: asim_test.cpp
19: $(CXX) $(CXXFLAGS) -c $<
20:
21: clean:
22: rm -f *.o *~ *.out
23:
24: distclean: clean
25: rm -f asim_test asim_test_gui
26:
```

Listing from directory: sdm/common

File: sdm/common/ErrorUtils.cpp

```
1: #include <unistd.h>
2: #include <stdio.h>
3: #include "ErrorUtils.h"
4:
5: void ErrorUtils::MemoryAllocError(const char* ErrorString)
6: {
7:     printf("Fatal Error - Could not allocate memory! \n Error: %s \n",ErrorString);
8:     _exit(-1);
9: }
10:
11: void ErrorUtils::MemoryAllocError()
12: {
13:     MemoryAllocError("");
14: }
15:
```

File: sdm/common/sdmLib.h

```
1: #ifndef _SDM_LIB_H_
2: #define _SDM_LIB_H_
3:
4: /* Misc. library and API macros*/
5:
6: #ifdef WIN32
7: // Function deprecation attributes, empty for a Windows build.
8: # define SDM_DEPRECATED
9: // The following ifdef block is the standard way of creating macros which make exporting
10: // from a DLL simpler. All files within this DLL are compiled with the SDMLIB_EXPORTS
11: // symbol defined on the command line. this symbol should not be defined on any project
12: // that uses this DLL. This way any other project whose source files include this file see
13: // SDMLIB_API functions as being imported from a DLL, whereas this DLL sees symbols
14: // defined with this macro as being exported.
15: # ifdef SDMLIB_EXPORTS
16: #   define SDMLIB_API __declspec(dllexport)
17: # else
18: #   define SDMLIB_API __declspec(dllimport)
19: # endif
20: #else
21: # define SDMLIB_API
22: # define SDM_DEPRECATED /*__attribute__((deprecated));*/
23: #endif
24:
25: #endif
```

File: sdm/common/version.h

```
1: #ifndef __SDM_VERSION_H__
2: #define __SDM_VERSION_H__
3:
4: #define REVISION_NUMBER 938
5: #define SDM_VERSION "1.10.3a 24MAY2010"
6:
7: #endif
```


File: sdm/common/UDPcom.h

```
1: #ifndef __UDPCOM_H_
2: #define __UDPCOM_H_
3:
4: #include <sys/types.h>
5:
6: #include "sdmLib.h"
7:
8: // UDPcom.h  UDP com library header
9:
10: #ifndef IP_SOCKET_INVALID
11: #define IP_SOCKET_INVALID (-1) // must match the definition in TCPcom.h
12: #endif
13: #ifndef UDP_SERV_RECV_SHUTDOWN
14: #define UDP_SERV_RECV_SHUTDOWN 0 //Returned from recvfrom when a socket has been
shutdown
15: #endif
16:
17: extern int SDMLIB_API UDPpassive (int port);           // setup server passive port
18:
19: extern int SDMLIB_API UDPserv_recv (int sock, void *buf, size_t length); // recv from anyone
(server passive socket)
20:
21: extern int SDMLIB_API UDPserv_reply (int sock, const void *buf, size_t length); // reply to last
message received on server
22:
23: extern int SDMLIB_API UDPserv_replyto (int sock, const void* buf, size_t length, const struct
sockaddr_in* s); // reply to some previous message recieved on server
24:
25: // send to (unconnected sock)
26: extern int SDMLIB_API UDPsendto (const char *ipaddr, int port, const void *buf, size_t length);
27:
28: extern int SDMLIB_API UDPrecvfrom (int port, void *buf, size_t length); // recv from anyone
29:
30: extern int SDMLIB_API UDPsend_broadcast(long bcast_addr, int port, const void *buf, size_t
length);
31:
32: extern int SDMLIB_API UDPconnect (const char *ipaddr, int port); // connected UDP socket for
UDPSend() & UDPrecv()
33: extern int SDMLIB_API UDPconnect (unsigned int pip, int port);
34:
```

```

35: extern int SDMLIB_API UDPsend (int sock, const void *buf, size_t length); // use with connected
UDP socket
36:
37: extern int SDMLIB_API UDPrecv (int sock, void *buf, size_t length); // use with connected UDP
socket
38:
39: extern void SDMLIB_API UDPgetip (struct sockaddr_in *p);    // get the ip of socket currently
connected
40:
41: extern int SDMLIB_API UDPavail (int sock, int timeout = 0); // return nonzero if a message is
42:                     // waiting on sock; wait for up to
43:                     // timeout milliseconds for a
44:                     // message to arrive
45:
46: extern int SDMLIB_API UDPset_rcv_timeout(int sock, int timeout);
47:
48: extern int SDMLIB_API UDPshutdown(int sock);
49:
50: extern int SDMLIB_API UDPclose(int sock);
51:
52: #endif

```

File: sdm/common/marshall.h

```
1: #ifndef __SDM_MARSHALL_H_
2: #define __SDM_MARSHALL_H_
3:
4: #ifndef __VXWORKS__
5: #include <endian.h> // Determine the byte order of the target CPU
6: #include <byteswap.h> //For byteswap macros
7: #else
8: #define bswap_32(x) ( ((0x000000FF&x) << 24) | ((0x0000FF00&x) << 8) | ((0x00FF0000&x) >> 8) |
((0xFF000000&x) >> 24) )
9: #define bswap_16(x) (((x) >> 8) & 0xff) | (((x) & 0xff) << 8))
10: #define bswap_64(x) (((x) & 0xff00000000000000ull) >> 56) \
11: | (((x) & 0x00ff000000000000ull) >> 40) \
12: | (((x) & 0x0000ff0000000000ull) >> 24) \
13: | (((x) & 0x000000ff00000000ull) >> 8) \
14: | (((x) & 0x00000000ff000000ull) << 8) \
15: | (((x) & 0x0000000000ff0000ull) << 24) \
16: | (((x) & 0x000000000000ff00ull) << 40) \
17: | (((x) & 0x00000000000000ffull) << 56))
18: #endif
19: /* SDM network byte order is little endian, which is the reverse of the
20: * traditional network order. Hence, these macros are nontrivial when
21: * ntohs(), htonl(), etc. are trivial and vice versa.
22: */
23:
24:
25:
26: #ifndef __uClinux__
27: #if __BYTE_ORDER == __LITTLE_ENDIAN
28: #define SDM_ntohs(x) (x)
29: #define SDM_htons(x) (x)
30: #define SDM_ntohl(x) (x)
31: #define SDM_htonl(x) (x)
32: #define SDM_ntohll(x) (x)
33: #define SDM_htonll(x) (x)
34: #else
35: #define SDM_ntohs(x) bswap_16 (x)
36: #define SDM_htons(x) bswap_16 (x)
37: #define SDM_ntohl(x) bswap_32 (x)
38: #define SDM_htonl(x) bswap_32 (x)
```

```

39: #define SDM_ntohll(x) bswap_64 (x)
40: #define SDM_htonll(x) bswap_64 (x)
41: #endif
42: #endif
43:
44: #ifndef __VXWORKS__
45: // Unmarshalling macros - generic
46: #define GET_TYPE1(type,ptr) (type)(*ptr)
47: #define GET_TYPE2(type,ptr) (type)(SDM_ntohs(*reinterpret_cast<const type *>(ptr)))
48: #define GET_TYPE4(type,ptr) (type)(SDM_ntohl(*reinterpret_cast<const type *>(ptr)))
49: #define GET_TYPE8(type,ptr) (type)(SDM_ntohll(*reinterpret_cast<const type *>(ptr)))
50:
51: // Unmarshalling macros - these assume a 32-bit architecture
52: #define GET_CHAR(ptr) GET_TYPE1 (char, ptr)
53: #define GET_UCHAR(ptr) GET_TYPE1 (unsigned char, ptr)
54: #define GET_SHORT(ptr) GET_TYPE2 (short, ptr)
55: #define GET_USHORT(ptr) GET_TYPE2 (unsigned short, ptr)
56: #define GET_INT(ptr) GET_TYPE4 (int, ptr)
57: #define GET_UINT(ptr) GET_TYPE4 (unsigned int, ptr)
58: #define GET_LONG(ptr) GET_TYPE4 (long, ptr)
59: #define GET_ULONG(ptr) GET_TYPE4 (unsigned long, ptr)
60: #define GET_FLOAT(ptr) GET_TYPE4 (float, ptr)
61: #define GET_DOUBLE(ptr) GET_TYPE8 (double, ptr)
62:
63: // Marshalling macros - generic
64: #define PUT_TYPE1(type,ptr,val) (*ptr) = ((type) val)
65: #define PUT_TYPE2(type,ptr,val) *reinterpret_cast<type *>(ptr) = SDM_htons(val)
66: #define PUT_TYPE4(type,ptr,val) *reinterpret_cast<type *>(ptr) = SDM_htonl(val)
67: #define PUT_TYPE8(type,ptr,val) *reinterpret_cast<type *>(ptr) = SDM_htonll(val)
68:
69: // Marshalling macros - these assume a 32-bit architecture
70: #define PUT_CHAR(ptr,val) PUT_TYPE1 (char,ptr, val)
71: #define PUT_UCHAR(ptr,val) PUT_TYPE1 (unsigned char,ptr, val)
72: #define PUT_SHORT(ptr,val) PUT_TYPE2 (short,ptr, val)
73: #define PUT_USHORT(ptr,val) PUT_TYPE2 (unsigned short,ptr, val)
74: #define PUT_INT(ptr,val) PUT_TYPE4 (int,ptr, val)
75: #define PUT_UINT(ptr,val) PUT_TYPE4 (unsigned int,ptr, val)
76: #define PUT_LONG(ptr,val) PUT_TYPE4 (long,ptr, val)
77: #define PUT_ULONG(ptr,val) PUT_TYPE4 (unsigned long,ptr, val)
78: #define PUT_FLOAT(ptr,val) PUT_TYPE4 (float,ptr, val)
79: #define PUT_DOUBLE(ptr,val) PUT_TYPE8 (double,ptr, val)

```

```

80:
81:
82: #elif defined(__uClinux__)
83:
84: #ifdef __cplusplus
85: extern "C" {
86: #endif
87: extern unsigned long long _marshall_get64(const void *pBuf);
88: extern double             _marshall_getdouble(const void *pBuf);
89: extern unsigned long      _marshall_get32(const void *pBuf);
90: extern float              _marshall_getfloat(const void *pBuf);
91: extern unsigned short     _marshall_get16(const void *pBuf);
92: extern unsigned char      _marshall_get8(const void *pBuf);
93: extern void               _marshall_put64(void *pBuf, unsigned long long iVal);
94: extern void               _marshall_putdouble(void *pBuf, double dVal);
95: extern void               _marshall_put32(void *pBuf, unsigned long iVal);
96: extern void               _marshall_putfloat(void *pBuf, float fVal);
97: extern void               _marshall_put16(void *pBuf, unsigned short iVal);
98: extern void               _marshall_put8(void *pBuf, unsigned char iVal);
99:
100: #define SDM_ntohs(x) (x)
101: #define SDM_htons(x) (x)
102: #define SDM_ntohl(x) (x)
103: #define SDM_htonl(x) (x)
104: #define SDM_ntohll(x) (x)
105: #define SDM_htonll(x) (x)
106:
107: // Unmarshalling macros - these assume a 32-bit architecture
108: #define GET_CHAR(ptr) ((char)      _marshall_get8(ptr))
109: #define GET_UCHAR(ptr) ((unsigned char) _marshall_get8(ptr))
110: #define GET_SHORT(ptr) ((short)     _marshall_get16(ptr))
111: #define GET_USHORT(ptr) ((unsigned short) _marshall_get16(ptr))
112: #define GET_INT(ptr) ((int)         _marshall_get32(ptr))
113: #define GET_UINT(ptr) ((unsigned int) _marshall_get32(ptr))
114: #define GET_LONG(ptr) ((long)       _marshall_get32(ptr))
115: #define GET_ULONG(ptr) ((unsigned long) _marshall_get32(ptr))
116: #define GET_FLOAT(ptr) ((float)     _marshall_getfloat(ptr))
117: #define GET_DOUBLE(ptr) ((double)   _marshall_getdouble(ptr))
118:
119:
120:

```

```

121: // Marshalling macros - these assume a 32-bit architecture
122: #define PUT_CHAR(ptr,val)  _marshall_put8(ptr, val)
123: #define PUT_UCHAR(ptr,val) _marshall_put8(ptr, val)
124: #define PUT_SHORT(ptr,val) _marshall_put16(ptr, val)
125: #define PUT_USHORT(ptr,val) _marshall_put16(ptr, val)
126: #define PUT_INT(ptr,val)   _marshall_put32(ptr, val)
127: #define PUT_UINT(ptr,val)  _marshall_put32(ptr, val)
128: #define PUT_LONG(ptr,val)  _marshall_put32(ptr, val)
129: #define PUT_ULONG(ptr,val) _marshall_put32(ptr, val)
130: #define PUT_FLOAT(ptr,val) _marshall_putfloat(ptr, val)
131: #define PUT_DOUBLE(ptr,val) _marshall_putdouble(ptr, val)
132:
133: #ifdef __cplusplus
134: }
135: #endif
136:
137: #else
138: #ifdef __cplusplus
139: extern "C" {
140: #endif
141: extern unsigned long long _marshall_get64(const void *pBuf);
142: extern double            _marshall_getdouble(const void *pBuf);
143: extern unsigned long     _marshall_get32(const void *pBuf);
144: extern float             _marshall_getfloat(const void *pBuf);
145: extern unsigned short    _marshall_get16(const void *pBuf);
146: extern unsigned char     _marshall_get8(const void *pBuf);
147: extern void              _marshall_put64(void *pBuf, unsigned long long iVal);
148: extern void              _marshall_putdouble(void *pBuf, double dVal);
149: extern void              _marshall_put32(void *pBuf, unsigned long iVal);
150: extern void              _marshall_putfloat(void *pBuf, float fVal);
151: extern void              _marshall_put16(void *pBuf, unsigned short iVal);
152: extern void              _marshall_put8(void *pBuf, unsigned char iVal);
153:
154: #define SDM_ntohs(x) (x)
155: #define SDM_htons(x) (x)
156: #define SDM_ntohl(x) (x)
157: #define SDM_htonl(x) (x)
158: #define SDM_ntohll(x) (x)
159: #define SDM_htonll(x) (x)
160:
161: // Unmarshalling macros - these assume a 32-bit architecture

```

```

162: #define GET_CHAR(ptr) ((char)      _marshall_get8(ptr))
163: #define GET_UCHAR(ptr) ((unsigned char) _marshall_get8(ptr))
164: #define GET_SHORT(ptr) ((short)      _marshall_get16(ptr))
165: #define GET_USHORT(ptr) ((unsigned short) _marshall_get16(ptr))
166: #define GET_INT(ptr) ((int)          _marshall_get32(ptr))
167: #define GET_UINT(ptr) ((unsigned int) _marshall_get32(ptr))
168: #define GET_LONG(ptr) ((long)        _marshall_get32(ptr))
169: #define GET_ULONG(ptr) ((unsigned long) _marshall_get32(ptr))
170: #define GET_FLOAT(ptr) ((float)       _marshall_getfloat(ptr))
171: #define GET_DOUBLE(ptr) ((double)     _marshall_getdouble(ptr))
172:
173:
174:
175: // Marshalling macros - these assume a 32-bit architecture
176: #define PUT_CHAR(ptr,val) _marshall_put8(ptr, val)
177: #define PUT_UCHAR(ptr,val) _marshall_put8(ptr, val)
178: #define PUT_SHORT(ptr,val) _marshall_put16(ptr, val)
179: #define PUT_USHORT(ptr,val) _marshall_put16(ptr, val)
180: #define PUT_INT(ptr,val) _marshall_put32(ptr, val)
181: #define PUT_UINT(ptr,val) _marshall_put32(ptr, val)
182: #define PUT_LONG(ptr,val) _marshall_put32(ptr, val)
183: #define PUT_ULONG(ptr,val) _marshall_put32(ptr, val)
184: #define PUT_FLOAT(ptr,val) _marshall_putfloat(ptr, val)
185: #define PUT_DOUBLE(ptr,val) _marshall_putdouble(ptr, val)
186:
187: #ifdef __cplusplus
188: }
189: #endif
190:
191: #endif
192:
193: #endif

```

File: sdm/common/UDPcom.cpp

```
1: // UDPcom.cpp  Cannon, UDP communications library
2:
3: #include <sys/types.h>
4: #include <sys/socket.h>
5: #ifndef __VXWORKS__
6: #include <sys/poll.h>
7: #endif
8: #include <netinet/in.h>
9: #include <arpa/inet.h>
10: #include <stdio.h>
11: #include <string.h>
12: #include <unistd.h>
13: #include <stdlib.h>
14: #include "sdmLib.h"
15: #include "marshall.h"
16: #include "UDPcom.h"
17:
18: #ifdef __VXWORKS__
19: #include <sockLib.h>
20: #include <selectLib.h>
21: #include <endian.h>
22: #endif
23:
24: #ifdef USE_SPACEWIRE
25: char zz[6];
26: #endif
27:
28: #ifndef WIN32
29: #define closesocket close
30: #endif
31:
32: struct sockaddr_in fsin;
33:
34:
35: SDMLIB_API
36: int UDPPassive (int port)           // setup server passive port
37: {  struct sockaddr_in sin;
38:     int sock;
39:     bool allow = true;
```



```

40:  memset (&sin, 0, sizeof(sin));
41:  sin.sin_family = AF_INET;
42:  sin.sin_addr.s_addr = INADDR_ANY;
43:  sin.sin_port = htons (static_cast<u_int16_t>(port));
44:  if( (sock = socket (PF_INET, SOCK_DGRAM, 0)) < 0)
45:  {
46:  #ifdef WIN32
47:      printf("UDPpassive::socket error %d \n", WSAGetLastError());
48:  #endif
49:  #ifndef WIN32
50:      perror("UDPpassive::socket error: ");
51:  #endif
52:      return IP_SOCKET_INVALID;
53:  }
54:  #ifdef __VXWORKS__
55:      setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,(char*)&allow,sizeof(allow));
56:  int bufSize = 25600;
57:  int sizeInt = 4;
58:  setsockopt(sock, SOL_SOCKET, SO_SNDBUF, (char*)&bufSize, sizeInt);
59:  #else
60:  setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,(const char*)&allow,sizeof(allow));
61:  #endif
62:  if (bind (sock, reinterpret_cast<struct sockaddr *>(&sin),sizeof(sin)) < 0)
63:  {  printf("Unable to bind the socket! (port:%d) \n",port);
64:  perror("UDPpassive");
65:      closesocket(sock);
66:      return IP_SOCKET_INVALID;
67:  }
68:
69:  int iRcvBufLen, optLen = sizeof(iRcvBufLen);
70:  #ifndef __VXWORKS__
71:  if (getsockopt(sock, SOL_SOCKET, SO_RCVBUF, (char*)&iRcvBufLen, (socklen_t*)&optLen) !=
-1)
72:  #else
73:  if (getsockopt(sock, SOL_SOCKET, SO_RCVBUF, (char*)&iRcvBufLen, (int*)&optLen) != -1)
74:  #endif
75:
76:  iRcvBufLen = 64 * 1024;
77:  #ifdef __VXWORKS__
78:  setsockopt(sock,(int)SOL_SOCKET,(int)SO_RCVBUF,(char*)&iRcvBufLen, sizeof(iRcvBufLen));
79:  #else

```

```

80: setsockopt(sock,(int)SOCKET,(int)SO_RCVBUF,(const char*)&iRcvBufLen,
sizeof(iRcvBufLen));
81: #endif
82: return sock;
83: }
84:
85:
86: SDMLIB_API
87: int UDPServ_rcv (int sock, void *buf, size_t length) // rcv from anyone (passive server socket)
88: {
89:     socklen_t alen = sizeof(fsin);
90: #ifndef __VXWORKS__
91:     return recvfrom (sock, (char*)buf, length, 0,
92:         reinterpret_cast<struct sockaddr *>(&fsin), (socklen_t*)&alen);
93: #else
94: return recvfrom (sock, (char*)buf, length, 0,
95:     reinterpret_cast<struct sockaddr *>(&fsin), (int*)&alen);
96: #endif
97: }
98:
99: SDMLIB_API
100: int UDPServ_reply (int sock, const void *buf, size_t length) // reply to last message received on
server
101: {
102:     return sendto (sock, (char*)buf, length, 0,
103:         reinterpret_cast<struct sockaddr *>(&fsin), sizeof(fsin));
104: }
105: SDMLIB_API
106: int UDPServ_replyto (int sock, const void* buf, size_t length, const struct sockaddr_in* s) // reply to
some previous message received on server
107: {
108:     struct sockaddr_in sin;
109:     memset(&sin,0,sizeof(sin));
110:     sin.sin_addr = s->sin_addr;
111:     sin.sin_port = static_cast<u_int16_t>(htons(s->sin_port));
112:     sin.sin_family = s->sin_family;
113:     return sendto (sock, (char*)buf, length, 0,
114:         reinterpret_cast<struct sockaddr *>(&sin), sizeof(sin));
115: }
116:
117: // send to (unconnected sock)
118: SDMLIB_API

```

```

119: int UDPsendto (const char *ipaddr, int port, const void *buf, size_t length)
120: { struct sockaddr_in sin;
121:   int sock, status;
122:   memset (&sin, 0, sizeof(sin));
123:   sin.sin_family = AF_INET;
124:   sin.sin_addr.s_addr = inet_addr ((char*)ipaddr);
125:   sin.sin_port = htons (static_cast<u_int16_t>(port));
126:   sock = socket (PF_INET, SOCK_DGRAM, 0);
127:   status = sendto (sock, (char*)buf, length, 0,
128:                   reinterpret_cast<struct sockaddr *>(&sin), sizeof(sin));
129:   if(status < 0)
130:   {
131:       perror("UDPsendto: ");
132:   }
133:   closesocket (sock);
134:   return status;
135: }
136:
137: SDMLIB_API
138: int UDPrecvfrom (int port, void *buf, size_t length) // recv from anyone (unconnected)
139: { struct sockaddr_in sin;
140:   int sock, status;
141:   memset (&sin, 0, sizeof(sin));
142:   sin.sin_family = AF_INET;
143:   sin.sin_addr.s_addr = INADDR_ANY;
144:   sin.sin_port = htons (static_cast<u_int16_t>(port));
145:   sock = socket (PF_INET, SOCK_DGRAM, 0);
146:   if (bind (sock, reinterpret_cast<struct sockaddr *>(&sin), sizeof(sin)) < 0)
147:   { perror ("UDPrecvfrom: ");
148:     return -1;
149:   }
150:   status = recv (sock, (char*)buf, length, 0);
151:   closesocket (sock);
152:   return status;
153: }
154:
155:
156: //bcast_addr must be in network byte order
157: SDMLIB_API
158: int UDPsend_broadcast(long bcast_addr, int port, const void *buf, size_t length)
159: {

```

```

160: int result;
161: int sock = -1;
162: struct sockaddr_in addr;
163: int bcast = 1;
164:
165: //Get a UDP socket
166: sock = socket(PF_INET,SOCK_DGRAM,0);
167: if (sock < 0)
168: {
169:     perror("Unable to get socket descriptor (SDMElection or SDMDMLeader): ");
170:     return sock;
171: }
172: #ifdef __VXWORKS__
173:     result = setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (char*)&bcast, sizeof(bcast));
174: #else
175:     result = setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (const char*)&bcast,
sizeof(bcast));
176: #endif
177:
178: if (result < 0)
179: {
180:     perror("Error in setsockopt: ");
181:     closesocket(sock);
182:     return result;
183: }
184: memset(&addr, 0, sizeof(addr));
185: addr.sin_family = AF_INET;
186: addr.sin_addr.s_addr = bcast_addr;
187: addr.sin_port = htons(port);
188:
189: result = sendto(sock, (char*) buf, length, 0,
190:     reinterpret_cast<struct sockaddr *>(&addr), sizeof(addr));
191: if (result < 0)
192: {
193:     perror("Unable to send broadcast message: ");
194:     closesocket(sock);
195:     return result;
196: }
197: // fprintf(stderr, "UDPSend_broadcast: sent %d to tmp socket, fd %d \n", length, sock);
198: closesocket(sock);
199: return length;

```

```

200: }
201:
202: static unsigned long spaceWireMask = 1;
203:
204: /* NOTE: pip must be in network byte order */
205: SDMLIB_API
206: int UDPconnect (unsigned int pip, int port)          // connected UDP socket for UDPsend() &
UDPrecv()
207: {
208:     struct sockaddr_in sin;          // pip is IP address in integer (long) form
209:     int sock;
210:     unsigned int uiSendIpAddr;
211:     int iSendPort;
212:
213:     memset (&sin, 0, sizeof(sin));
214:     sin.sin_family = AF_INET;
215:     uiSendIpAddr = pip;
216:     iSendPort = port;
217:
218: #ifdef USE_SPACEWIRE
219:     // Get the Spacewire netmask
220:     // if ( spaceWireMask == 1 )
221:     // {
222:         // char* netmaskStr = getenv("SpaceWireNetMask");
223:         // if ( netmaskStr != NULL )
224:         // {
225:             // spaceWireMask = inet_addr(netmaskStr);
226: // #ifdef WIN32
227:             // if ( spaceWireMask == INADDR_NONE )
228: // #else
229:             // if ( spaceWireMask == (in_addr_t)(-1)) )
230: // #endif
231:             // spaceWireMask = 0;
232:         // }
233:         // else
234:             // spaceWireMask = 0;
235:         // printf("using SpaceWireMask %08x \n",spaceWireMask);
236:     // }
237:     // if((pip&spaceWireMask)==pip)
238:     // If the IP address begins with 10., then forward it to the NM bridge agent on localhost
239:     // to be sent out the SpaceWire interface.

```

```

240: #ifdef __LITTLE_ENDIAN
241:     if((pip&0xff)== 0xa)
242: #else
243:     if((bswap_32(pip)&0xff)== 0xa)
244: #endif
245:     {
246:
247:         // Warning: this is not thread safe if multiple threads are sending messages.
248:         // Nor is it safe for opening multiple sockets using UDPconnect().
249:         // This is probably OK for testing purposes in the RST, but a more robust solution
250:         // should be devised if problems are suspected.
251:
252:         zz[3] = (pip>>24)&0xff; zz[2] = (pip>>16)&0xff; zz[1] = (pip>>8)&0xff; zz[0] = pip &
0xff;
253:         zz[4] = ((port&0xff00)>>8); zz[5] = (port&0xff);
254:         //printf("UDPconnect..addr=%x, zz=%x,%x,%x,%x \n",pip,zz[0],zz[1],zz[2],zz[3]);
255:         uiSendIpAddr = inet_addr ("127.0.0.1");    // ipaddr is in number-dot notation
256:         iSendPort = 7680;
257:     }
258:     else
259:     {
260:         zz[0]=0;
261:     }
262: #endif
263:
264:     sin.sin_port = htons (static_cast<u_int16_t>(iSendPort));
265:     sin.sin_addr.s_addr = uiSendIpAddr;
266:     sock = socket (PF_INET, SOCK_DGRAM, 0);
267:
268:     int iSndBufLen, optLen = sizeof(iSndBufLen);
269:     iSndBufLen = 25 * 1024;
270:
271: #ifdef __VXWORKS__
272:     setsockopt(sock, SOL_SOCKET, SO_SNDBUF, (char*)&iSndBufLen, sizeof(iSndBufLen));
273: #else
274:     setsockopt(sock, SOL_SOCKET, SO_SNDBUF, (const char*)&iSndBufLen,
sizeof(iSndBufLen));
275: #endif
276:
277:     if(connect (sock, reinterpret_cast<struct sockaddr *>(&sin),sizeof(sin)) < 0)
278:     {

```

```

279:     struct in_addr in = {uiSendIpAddr};
280:
281:     printf("Unable to connect to %s:%d \n",inet_ntoa(in), iSendPort);
282:     perror("UDPconnect");
283:     return IP_SOCKET_INVALID;
284: }
285: return sock;
286: }
287:
288: SDMLIB_API
289: int UDPconnect (const char *ipaddr, int port)    // connected UDP socket for UDPsend() &
UDPrecv()
290: {
291:     int sock;
292:     in_addr_t pip = inet_addr ((char*)ipaddr);    // ipaddr is in number-dot notation
293:     sock = UDPconnect (pip, port);
294:     if(sock == IP_SOCKET_INVALID)
295:     {
296:         printf("Failed to connect to %s:%d \n",ipaddr,port);
297:     }
298:     return sock;
299: }
300:
301: SDMLIB_API
302: int UDPsend (int sock, const void *buf, size_t length) // use with connected UDP socket
303: {
304: #ifdef USE_SPACEWIRE
305:     //
306:     // SPA-S specific code -- If this socket is associated with a SpaceWire bus address,
307:     // then send the message to the local NM bridge agent.
308:     if(zz[0] != 0)
309:     {
310:         unsigned int i = length;
311:         i += 6;
312:         char* buf1 = (char*)malloc(i);
313:         memcpy(buf1,zz,6); memcpy(&buf1[6],buf,i-6);
314:         int result = send(sock,buf1,i,0);
315:         //printf("UDPSend..result=%x,error=%d \n",result,WSAGetLastError());
316:         free(buf1);
317:         return result;
318:     }

```

```

319:  //
320:  // End Spacewire specific code
321: #endif
322:  int sendResult = send (sock, (char*)buf, length, 0);
323:  return sendResult;
324: }
325:
326: SDMLIB_API
327: int UDPrecv (int sock, void *buf, size_t length) // use with connected UDP socket
328: {  return recv (sock, (char*)buf, length, 0);
329: }
330:
331: SDMLIB_API
332: void UDPgetip (struct sockaddr_in *p)
333: {
334:   p->sin_addr = fsin.sin_addr;
335:   p->sin_port = ntohs(static_cast<u_int16_t>(fsin.sin_port));
336:   p->sin_family = fsin.sin_family;
337: }
338:
339: /* NOTE: timeout is in milliseconds */
340: SDMLIB_API
341: int UDPavail (int sock, int timeout)
342: {
343: #ifndef __VXWORKS__
344:   struct pollfd ufd;
345:   ufd.fd = sock;
346:   ufd.events = POLLIN | POLLPRI;
347:   return poll(&ufd, 1U, timeout) > 0;
348: #else
349:   struct fd_set fds;
350:   FD_SET(sock, &fds);
351:   timeval timeoutVal;
352:   timeoutVal.tv_usec = timeout;
353:   return (select(sock + 1, &fds, NULL, NULL, &timeoutVal) > 0);
354: #endif
355: }
356:
357: /* NOTE: timeout is in milliseconds */
358: SDMLIB_API
359: int UDPset_rcv_timeout (int sock, int timeout)

```



```

360: {
361:     int result;
362:
363: #ifdef WIN32
364:     // timeout is already in milliseconds
365:     result = setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
366: #else
367:     struct timeval time;
368:     time.tv_sec = timeout / 1000;
369:     time.tv_usec = (timeout % 1000) * 1000;
370:     result = setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char*)&time, sizeof(time));
371: #endif
372:
373:     return result;
374: }
375:
376: SDMLIB_API
377: int UDPshutdown (int sock)
378: {
379: #ifndef WIN32
380:     return shutdown(sock, SHUT_RDWR);
381: #else
382:     return shutdown(sock, SD_BOTH);
383: #endif
384: }
385:
386: SDMLIB_API
387: int UDPclose (int sock)
388: {
389:     return closesocket(sock);
390: }

```

File: sdm/common/sdmLib.cpp

```
1: // sdmLib.cpp : Defines the entry point for the DLL application.
2: //
3:
4: #include "sdmLib.h"
5: #include "Time/TimeKeeping.h"
6:
7:
8: #ifdef WIN32
9: #include <winsock2.h>
10: BOOL APIENTRY DllMain( HANDLE hModule,
11:     DWORD ul_reason_for_call,
12:     LPVOID lpReserved
13:     )
14: {
15:     unsigned short wVersionRequested;
16:     WSADATA wsaData;
17:     int err;
18:     switch (ul_reason_for_call)
19:     {
20:     case DLL_PROCESS_ATTACH:
21:         InitializeTimeKeeping();
22:         wVersionRequested = MAKEWORD( 2, 2 );
23:         err = WSAStartup( wVersionRequested, &wsaData );
24:         if ( err != 0 ) {
25:             // Tell the user that we could not find a usable WinSock DLL.
26:             return -1;
27:         }
28:         break;
29:     case DLL_THREAD_ATTACH:
30:         break;
31:     case DLL_THREAD_DETACH:
32:         break;
33:     case DLL_PROCESS_DETACH:
34:         WSACleanup();
35:         break;
36:     }
37:     return TRUE;
38: }
39:
```

```

40: // This is an example of an exported variable
41: SDMLIB_API int nsdmLib=0;
42:
43: // This is an example of an exported function.
44: SDMLIB_API int fnsdmLib(void)
45: {
46: return 42;
47: }
48:
49: // This is the constructor of a class that has been exported.
50: // see sdmLib.h for the class definition
51: //CsdmLib::CsdmLib()
52: //{
53: //    return;
54: //}
55:
56: #else
57: extern SDMLIB_API void InitializeTimeKeeping();
58:
59: void __attribute__((constructor)) my_init(void)
60: {
61: InitializeTimeKeeping();
62: }
63:
64:
65: void __attribute__((destructor)) my_fini(void)
66: {
67:
68: }
69:
70: #endif

```

File: sdm/common/marshall.c

```
1: #include <byteswap.h>
2: #include <string.h>
3: #include <stdio.h>
4:
5: #define BSWAP_2(d, s) ((char*)(d))[0] = ((char*)(s))[1], ((char*)(d))[1] = ((char*)(s))[0]
6: #define BSWAP_4(d, s) ((char*)(d))[0] = ((char*)(s))[3], ((char*)(d))[1] = ((char*)(s))[2],
((char*)(d))[2] = ((char*)(s))[1], ((char*)(d))[3] = ((char*)(s))[0]
7: #define BSWAP_8(d, s) ((char*)(d))[0] = ((char*)(s))[7], ((char*)(d))[1] = ((char*)(s))[6],
((char*)(d))[2] = ((char*)(s))[5], ((char*)(d))[3] = ((char*)(s))[4], ((char*)(d))[4] = ((char*)(s))[3],
((char*)(d))[5] = ((char*)(s))[2], ((char*)(d))[6] = ((char*)(s))[1], ((char*)(d))[7] = ((char*)(s))[0]
8:
9: unsigned long long _marshall_get64(const void *pBuf)
10: {
11:     unsigned long long i;
12:     BSWAP_8(&i, pBuf);
13:
14:     return i;
15: }
16:
17: double _marshall_getdouble(const void *pBuf)
18: {
19:     double d;
20:     BSWAP_8(&d, pBuf);
21:
22:     return d;
23: }
24:
25:
26: unsigned long _marshall_get32(const void *pBuf)
27: {
28:     unsigned long i;
29:     BSWAP_4(&i, pBuf);
30:
31:     return i;
32: }
33:
34:
35: float _marshall_getfloat(const void *pBuf)
36: {
37:     float f;
```

```

38: BSWAP_4(&f, pBuf);
39:
40: return f;
41: }
42:
43:
44: unsigned short _marshall_get16(const void *pBuf)
45: {
46:     unsigned short i;
47:     BSWAP_2(&i, pBuf);
48:
49:     return i;
50: }
51:
52: unsigned char _marshall_get8(const void *pBuf)
53: {
54:     return *(unsigned char*)(pBuf);
55: }
56:
57: void _marshall_put64(void *pBuf, unsigned long long iVal)
58: {
59:     BSWAP_8(pBuf, &iVal);
60: }
61:
62: void _marshall_putdouble(void *pBuf, double dVal)
63: {
64:     BSWAP_8(pBuf, &dVal);
65: }
66:
67: void _marshall_put32(void *pBuf, unsigned long iVal)
68: {
69:     BSWAP_4(pBuf, &iVal);
70: }
71:
72: void _marshall_putfloat(void *pBuf, float fVal)
73: {
74:     BSWAP_4(pBuf, &fVal);
75: }
76:
77: void _marshall_put16(void *pBuf, unsigned short iVal)
78: {

```

```
79: BSWAP_2(pBuf, &iVal);
80: }
81:
82: void _marshall_put8(void *pBuf, unsigned char iVal)
83: {
84:     *(unsigned char*)pBuf = iVal;
85: }
```

File: sdm/common/TCPcom.h

```
1: #ifndef __TCPCOM_H_
2: #define __TCPCOM_H_
3:
4: #include <sys/types.h>
5:
6: #include "sdmLib.h"
7:
8: // TCPcom.h TCP com library header
9:
10: #ifndef IP_SOCKET_INVALID
11: #define IP_SOCKET_INVALID (-1) // must match the definition in UDPcom.h
12: #endif
13: #ifndef TCP_RECV_SHUTDOWN
14: #define TCP_RECV_SHUTDOWN 0 //Returned from recvfrom when a socket has been shutdown
15: #endif
16:
17: extern int SDMLIB_API TCPpassive (int port, int maxConn); // setup server passive
port
18:
19: extern int SDMLIB_API TCPaccept (int sock, struct sockaddr_in *sin); // accept a connection
20:
21: extern int SDMLIB_API TCPserv_accept (int sock); // accept a connection
22:
23: extern int SDMLIB_API TCPserv_recv (int sock, void *buf, size_t length); // recv from anyone
(server passive socket)
24:
25: extern int SDMLIB_API TCPserv_reply (int sock, const void *buf, size_t length); // reply to last
message received on server
26:
27: extern int SDMLIB_API TCPserv_replyto (int sock, const void* buf, size_t length, const struct
sockaddr_in* s); // reply to some previous message recieved on server
28:
29: // send to (unconnected sock)
30: extern int SDMLIB_API TCPsendto (const char *ipaddr, int port, const void *buf, size_t length);
31:
32: extern int SDMLIB_API TCPrecvfrom (int port, void *buf, size_t length); // recv from anyone
33:
34: extern int SDMLIB_API TCPconnect (const char *ipaddr, int port); // connected TCP socket for
TCPsend() & TCPrecv()
35: extern int SDMLIB_API TCPconnect (unsigned int pip, int port);
```

```

36:
37: extern int SDMLIB_API TCPsend (int sock, const void *buf, size_t length); // use with connected
TCP socket
38:
39: extern int SDMLIB_API TCPrecv (int sock, void *buf, size_t length); // use with connected TCP
socket
40:
41: extern void SDMLIB_API TCPgetip (struct sockaddr_in *p); // get the ip of socket currently
connected
42:
43: extern int SDMLIB_API TCPavail (int sock, int timeout = 0); // return nonzero if a message is
44:             // waiting on sock; wait for up to
45:             // timeout milliseconds for a
46:             // message to arrive
47:
48: extern int SDMLIB_API TCPshutdown(int sock);
49:
50: extern int SDMLIB_API TCPclose(int sock);
51:
52: #endif

```


File: sdm/common/SDMComHandle.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMComHandle.h"
4: #include "TCPcom.h"
5:
6: void CopySockAddrIn(struct sockaddr_in* Target, const struct sockaddr_in* Source)
7: {
8:     memset(Target, 0, sizeof(struct sockaddr_in));
9:     if (Source != NULL)
10:    {
11:        Target->sin_family = Source->sin_family;
12:        Target->sin_port = Source->sin_port;
13:        Target->sin_addr.s_addr = Source->sin_addr.s_addr;
14:    }
15: }
16:
17: SDMComHandle::SDMComHandle()    :    m_iSock(IP_SOCKET_INVALID),    m_bIsTcp(false),
m_sinAddress()
18: {
19:     memset(&m_sinAddress, 0, sizeof(struct sockaddr_in));
20: }
21:
22: // Destructor does not cleanup the com handle, this should be called specifically when the client
23: // is finished using it
24: SDMComHandle::~SDMComHandle()
25: {
26: }
27:
28: SDMComHandle::SDMComHandle(int Sock, bool TCP, const struct sockaddr_in* Address) :
m_iSock(Sock), m_bIsTcp(TCP), m_sinAddress()
29: {
30:     CopySockAddrIn(&m_sinAddress, Address);
31: }
32:
33: SDMComHandle::SDMComHandle(const SDMComHandle& right) : m_iSock(right.m_iSock),
m_bIsTcp(right.m_bIsTcp), m_sinAddress()
34: {
35:     CopySockAddrIn(&m_sinAddress, &right.m_sinAddress);
36: }
37:
```

```

38: SDMComHandle& SDMComHandle::operator=(const SDMComHandle& right)
39: {
40:     m_iSock = right.m_iSock;
41:     m_bIsTcp = right.m_bIsTcp;
42:     CopySockAddrIn(&m_sinAddress, &right.m_sinAddress);
43:
44:     return *this;
45: }
46:
47: void SDMComHandle::Set(int Sock, bool Tcp, const sockaddr_in* Address)
48: {
49:     m_iSock = Sock;
50:     m_bIsTcp = Tcp;
51:     CopySockAddrIn(&m_sinAddress, Address);
52: }
53:
54: void SDMComHandle::DoCleanup()
55: {
56:     // Only close the ComHandle for TCP sockets
57:     if (m_bIsTcp && m_iSock != IP_SOCKET_INVALID)
58:     {
59:         TCPclose(m_iSock);
60:         m_iSock = IP_SOCKET_INVALID;
61:         m_bIsTcp = false;
62:     }
63: }

```

File: sdm/common/SDMCancelQueue.cpp

```
1: #include "SDMCancelQueue.h"
2:
3: void SDMCancelQueue::add(xTEDSPParameters* toAdd)
4: {
5:     if(tailIndex + 1 == headIndex || (tailIndex + 1 == SIZE - 1 && headIndex == 0))
6:     {
7:         printf("Queue is full \n");
8:         return;
9:     }
10:
11:     if(tailIndex == SIZE - 1)
12:     {
13:         tailIndex = 0;
14:     }
15:     queue[tailIndex] = toAdd;
16:     tailIndex++;
17: }
18:
19:
20: xTEDSPParameters* SDMCancelQueue::dequeue(void)
21: {
22:     if(headIndex == tailIndex)
23:     {
24:         printf("Queue is empty \n");
25:         return NULL;
26:     }
27:
28:     xTEDSPParameters* toReturn = queue[headIndex];
29:     headIndex++;
30:     if(headIndex == SIZE - 1)
31:     {
32:         headIndex = 0;
33:     }
34:     return toReturn;
35: }
36:
37:
38: int SDMCancelQueue::size(void)
39: {
```

```
40: int size = 0;
41: if(headIndex == tailIndex)
42: {
43:     size = 0;
44: }
45: else if(headIndex < tailIndex)
46: {
47:     size = tailIndex - headIndex;
48: }
49: else
50: {
51:     size = (SIZE - headIndex) + tailIndex;
52: }
53:
54: return size;
55: }
```

File: sdm/common/SDMRegQueue.h

```
1: #ifndef SDM_REG_QUEUE_H
2: #define SDM_REG_QUEUE_H
3:
4: #include <stdio.h>
5: #include <string.h>
6: #include "message/SDMComponent_ID.h"
7:
8: #define SIZE 100
9:
10: //Used to queue up incoming requests to register with the SDM
11: class SDMLIB_API SDMRegQueue
12: {
13: public:
14:     SDMRegQueue(): headIndex(0), tailIndex(0)
15:     {
16:     }
17:
18:     void add(SDMComponent_ID toAdd);
19:     SDMComponent_ID dequeue(void);
20:     int size(void);
21:     bool find(const SDMComponent_ID& toFind);
22:
23: private:
24:     SDMComponent_ID queue[SIZE];
25:     int headIndex;
26:     int tailIndex;
27: };
28:
29: #endif
```

File: sdm/common/SDMRegQueue.cpp

```
1: #include "SDMRegQueue.h"
2:
3: void SDMRegQueue::add(SDMComponent_ID toAdd)
4: {
5:     if(tailIndex + 1 == headIndex || (tailIndex + 1 == SIZE - 1 && headIndex == 0))
6:     {
7:         printf("Queue is full \n");
8:         return;
9:     }
10:
11:     if(tailIndex == SIZE - 1)
12:     {
13:         tailIndex = 0;
14:     }
15:     queue[tailIndex] = toAdd;
16:     tailIndex++;
17: }
18:
19:
20: SDMComponent_ID SDMRegQueue::dequeue(void)
21: {
22:     if(headIndex == tailIndex)
23:     {
24:         printf("Queue is empty \n");
25:         SDMComponent_ID empty;
26:         return empty;
27:     }
28:
29:     SDMComponent_ID toReturn = queue[headIndex];
30:     headIndex++;
31:     if(headIndex == SIZE - 1)
32:     {
33:         headIndex = 0;
34:     }
35:     return toReturn;
36: }
37:
38: int SDMRegQueue::size(void)
39: {
```

```

40: int size = 0;
41: if(headIndex == tailIndex)
42: {
43:     size = 0;
44: }
45: else if(headIndex < tailIndex)
46: {
47:     size = tailIndex - headIndex;
48: }
49: else
50: {
51:     size = (SIZE - headIndex) + tailIndex;
52: }
53:
54: return size;
55: }
56:
57: bool SDMRegQueue::find(const SDMComponent_ID& toFind)
58: {
59:     int first = 0;
60:
61:     if(headIndex >= 0)
62:     {
63:         first = headIndex;
64:     }
65:
66:     if(headIndex < tailIndex) //Check queue, no wrap
67:     {
68:         for(int i = first; i < tailIndex; i++)
69:         {
70:             if(queue[i] == toFind)
71:             {
72:                 return true;
73:             }
74:         }
75:     }
76:     else if(tailIndex < headIndex) //If you have to wrap around
77:     {
78:         for(int i = first; i < SIZE; i++)
79:         {
80:             if(queue[i] == toFind)

```

```
81:    {
82:        return true;
83:    }
84: }
85: for(int i = 0; i < tailIndex; i++)
86: {
87:     if(queue[i] == toFind)
88:     {
89:         return true;
90:     }
91: }
92: }
93: return false;
94: }
95:
```


File: sdm/common/MemoryUtils.h

```
1: #ifndef _SDM_MEMORY_UTILS_H_
2: #define _SDM_MEMORY_UTILS_H_
3:
4: #include <string.h>
5: #include "sdmLib.h"
6:
7: extern SDMLIB_API char* SDM_strdup(const char* s);
8: extern SDMLIB_API char* SDM_strndup(const char* s, size_t n);
9: extern SDMLIB_API void* SDM_malloc(size_t size);
10: extern SDMLIB_API void SDMMemoryAllocError(const char* ErrorString);
11: #ifdef __uCLinux__
12: char* strdup(const char* str, size_t n);
13: #endif
14:
15: #ifdef __VXWORKS__
16: char* strdup(const char* str, size_t n);
17: #endif
18:
19: #endif
```

File: sdm/common/Makefile

```
1: # Common code Makefile, produces a shared object
2: include ../Makefile.common
3: include ../$(MAKEFILE_DEFS)
4:
5: .PHONY:    all clean distclean
6:
7: SUBDIRS=message MessageLogger MessageManager MessageManipulator SubscriptionManager
task checksum asim semaphore VarInfoParser xTEDS Time Exception Regex
8:
9: MessageFiles = SDMAck SDMCancel SDMCancelxTEDS SDMCode SDMCommand SDMConsume
SDMData SDMDeletesub SDMDMLeader SDMElection SDMError SDMHeartbeat SDMmessage
SDMPostTask SDMReady SDMRegInfo SDMReqCode SDMReqReg SDMReqxTEDS SDMSerreqst
SDMService SDMSubreqst SDMTask SDMTaskError SDMTaskFinished SDMTat SDMxTEDS
SDMxTEDSInfo SDMRegPM SDMComponent_ID SDMMessage_ID SDMSearch SDMSearchReply
SDMVarInfo SDMVarReq SDKill SDMHello SDMRegister SDMID
10:
11: MessageLoggerFiles = MessageLogger SDMMMessageLogger
12:
13: MessageManagerFiles = MessageManager
14:
15: MessageManipulatorFiles = MessageManipulator lex.MessageManipulator msgdef.tab message
16:
17: SubscriptionManagerFiles = SubscriptionManager
18:
19: TaskFiles = SDMTaskResources
20:
21: TimeFiles = TimeKeepingLinux SDMTimeLinux TimerList SecTime
22:
23: ChecksumFiles = checksum crcmodel
24:
25: AsimFiles = ASIM
26:
27: SemaphoreFiles = semaphore
28:
29: VarInfoParserFiles = Variable lex.VarInfoParser VarInfoParser.tab VarInfoParser
30:
31: xTEDSFiles = lex.xTEDS MessageDef xTEDSCommand xTEDSDataMsg xTEDSItem
xTEDSItemTree xTEDSMessage xTEDS xTEDSParser xTEDSQualifierList xTEDSQualifier xTEDS.tab
xTEDSVariable xTEDSRequest xTEDSNotification xTEDSCommandMsg xTEDSFaultMsg
xTEDSDrange xTEDSOption xTEDSOptionList xTEDSOrientationList xTEDSOrientationItem
```

```

xTEDSCurve xTEDSCoef xTEDSCoefList xTEDSLocation VariableDef xTEDSVerification
xTEDSWrapper xTEDSWrapperList xTEDSVariableList
32:
33: ExceptionFiles = SDMException SDMRegexException SDMBadIndexException
34:
35: RegexFiles = Regex RegexResult RegexMatch RegexCapture RegularExpression
36:
37: BUILDTARGETS=$(addsuffix .o, \
38:     $(addprefix ./message/, $(MessageFiles)) \
39:     $(addprefix ./MessageLogger/, $(MessageLoggerFiles)) \
40:     $(addprefix ./MessageManager/, $(MessageManagerFiles)) \
41:     $(addprefix ./MessageManipulator/, $(MessageManipulatorFiles)) \
42:     $(addprefix ./SubscriptionManager/, $(SubscriptionManagerFiles)) \
43:     $(addprefix ./task/, $(TaskFiles)) \
44:     $(addprefix ./Time/, $(TimeFiles)) \
45:     $(addprefix ./checksum/, $(ChecksumFiles)) \
46:     $(addprefix ./asim/, $(AsimFiles)) \
47:     $(addprefix ./semaphore/, $(SemaphoreFiles)) \
48:     $(addprefix ./VarInfoParser/, $(VarInfoParserFiles)) \
49:     $(addprefix ./xTEDS/, $(xTEDSFiles)) \
50:     $(addprefix ./Exception/, $(ExceptionFiles)) \
51:     $(addprefix ./Regex/, $(RegexFiles))) \
52:     UDPcom.o TCPcom.o ErrorUtils.o MemoryUtils.o SDMComHandle.o SDMRegQueue.o
SDMCancelQueue.o sdmLib.o
53:
54: all: subdir libSDM.so libSDM.so.1 libSDM.so.1.0 libSDM.a
55:
56: uclinux: subdir libSDM.a
57:
58: libSDM.so: libSDM.so.1.0
59: ln -sf $< $@
60:
61: libSDM.so.1: libSDM.so.1.0
62: ln -sf $< $@
63:
64: libSDM.so.1.0: $(BUILDTARGETS)
65: $(CXX) $(CXXFLAGS) -fPIC -shared -o $@ $^
66:
67: libSDM.a: $(BUILDTARGETS) marshall.o
68: ar rcs $@ $^
69:

```

```

70: subdir:
71: for dir in $(SUBDIRS); do \
72:     make -C $$dir; \
73: done
74:
75: UDPcom.o: UDPcom.cpp UDPcom.h
76: $(CXX) $(CXXFLAGS) -fPIC -c $<
77:
78: TCPcom.o: TCPcom.cpp TCPcom.h
79: $(CXX) $(CXXFLAGS) -fPIC -c $<
80:
81: ErrorUtils.o: ErrorUtils.cpp ErrorUtils.h
82: $(CXX) $(CXXFLAGS) -fPIC -c $<
83:
84: MemoryUtils.o: MemoryUtils.c MemoryUtils.h
85: $(CC) $(CFLAGS) -fPIC -c $<
86:
87: SDMComHandle.o: SDMComHandle.cpp SDMComHandle.h
88: $(CXX) $(CXXFLAGS) -fPIC -c $<
89:
90: SDMRegQueue.o: SDMRegQueue.cpp SDMRegQueue.h
91: $(CXX) $(CXXFLAGS) -fPIC -c $<
92:
93: SDMCancelQueue.o: SDMCancelQueue.cpp SDMCancelQueue.h
94: $(CXX) $(CXXFLAGS) -fPIC -c $<
95:
96: sdmLib.o: sdmLib.cpp
97: $(CXX) $(CXXFLAGS) -I./ -fPIC -c $<
98:
99: marshall.o: marshall.c marshall.h
100: $(CC) $(CCFLAGS) -fPIC -c $<
101:
102: clean:
103:     for dir in $(SUBDIRS); do \
104:         make -C $$dir clean; \
105:     done
106:     rm -f *.o *~
107:
108: distclean: clean
109:     for dir in $(SUBDIRS); do \
110:         make -C $$dir distclean; \

```

```
111: done
112: rm -f libSDM.so.* libSDM.so libSDM.a
```

File: sdm/common/MemoryUtils.c

```
1: #include <stdlib.h>
2: #include <stdio.h>
3: #include <unistd.h>
4: #include <string.h>
5: #include "MemoryUtils.h"
6:
7: SDMLIB_API
8: char* SDM_strdup(const char* s)
9: {
10: char* String = strdup(s);
11: if (String == NULL)
12: {
13:     SDMMemoryAllocError(__FUNCTION__);
14: }
15: return String;
16: }
17:
18: SDMLIB_API
19: char* SDM_strndup(const char* s, size_t n)
20: {
21: char* String = strndup(s, n);
22:
23: if (String == NULL)
24: {
25:     SDMMemoryAllocError(__FUNCTION__);
26: }
27: return String;
28: }
29:
30: SDMLIB_API
31: void* SDM_malloc(size_t size)
32: {
33: void* Memory = malloc(size);
34: if (Memory == NULL)
35: {
36:     SDMMemoryAllocError(__FUNCTION__);
37: }
38: return Memory;
39: }
```

```

40:
41: SDMLIB_API
42: void SDMMemoryAllocError(const char* ErrorString)
43: {
44:     printf("Fatal Error - Could not allocate memory! \n Error: %s \n", ErrorString);
45:     _exit(-1);
46: }
47:
48: #ifdef __uClinux__
49: /* Petalinux for the ucLinux build doesn't have strndup */
50: char* strndup(const char* str, size_t n)
51: {
52:     size_t len = strlen(str);
53:     if ( len > n )
54:     {
55:         char* dup = (char*)malloc(n+1);
56:         strncpy(dup,str,n);
57:         dup[n] = 0;
58:         return dup;
59:     }
60:     else
61:         return strdup(str);
62: }
63: #endif
64:
65: #ifdef __VXWORKS__
66: /* VxWorks doesn't have strndup */
67: char* strndup(const char* str, size_t n)
68: {
69:     size_t len = strlen(str);
70:     if ( len > n )
71:     {
72:         char* dup = (char*)malloc(n+1);
73:         strncpy(dup,str,n);
74:         dup[n] = 0;
75:         return dup;
76:     }
77:     else
78:         return strdup(str);
79: }
80: #endif

```

81:
82:
83:

File: sdm/common/SDMError.h

File: sdm/common/asensor.h

```
1: #ifndef __ASENSOR_H_
2: #define __ASENSOR_H_
3:
4: #define    SENSOR_ANALOG_SOURCE 0x0001
5: #define    SENSOR_THERMOCOUPLE  0x0002
6: #define    SENSOR_VGI_AMP        0x0003
7: #define    SENSOR_DIGITAL_IO     0x0004
8:
9: #define    SENSOR_PRODUCT_ID    0xC001
10: #define    SENSOR_TIMEOUT_SEC  0xC002
11:
12: #define    VGI_AMP_STAGE_TWO_GAIN  0x01
13: #define    VGI_AMP_STAGE_ONE_GAIN  0x02
14: #define    VGI_AMP_OFFSET          0x03
15:
16: #define    ANALOG_SOURCE_DAC_OUT  0x01
17:
18: #define    DIGITAL_IO_PROGRAM 0x01
19: #define    DIGITAL_IO_IO      0x02
20:
21: #define    SENSOR_WRITE_FLASH 0xFA
22: #define    SENSOR_ERASE_FLASH 0xFB
23: #define    SENSOR_READB_FLASH 0xFC
24: #define    SENSOR_READW_FLASH  0xFD
25:
26: #endif
27:
```

File: sdm/common/SDMCancelQueue.h

```
1: #ifndef SDM_CANCEL_QUEUE_H
2: #define SDM_CANCEL_QUEUE_H
3:
4: #include <stdio.h>
5: #include <string.h>
6: #include "../dm/xTEDSPParameters.h"
7:
8: #define SIZE 100
9:
10: //Used to queue up incoming xTEDS cancelations
11: class SDMLIB_API SDMCancelQueue
12: {
13: public:
14:     SDMCancelQueue(): headIndex(0), tailIndex(0)
15:     {
16:     }
17:
18:     void add(xTEDSPParameters* toAdd);
19:     xTEDSPParameters* dequeue(void);
20:     int size(void);
21:
22: private:
23:     xTEDSPParameters* queue[SIZE];
24:     int headIndex;
25:     int tailIndex;
26: };
27:
28: #endif
```

File: sdm/common/Debug.h

```
1: #ifndef __SDM_DEBUG_H_
2: #define __SDM_DEBUG_H_
3:
4: // A macro for debug output, which can be conditionally removed, the variable "debug"
5: // must be defined at global scope, as in the DM, TM, SM, and PM. Called debug_f because
6: // all params beyond "Level" are passed to printf.
7: #ifndef REMOVE_DEBUG_OUTPUT
8: # ifndef debug_f
9: #  define debug_f(Level,...) do { if (Level<=debug) { printf(__VA_ARGS__); } } while(0)
10: # endif
11: #else
12: #  define debug_f(Level,...)
13: #endif
14:
15: #endif //ifndef __SDM_DEBUG_H_
```

File: sdm/common/TCPcom.cpp

```
1: // TCPcom.cpp  TCP communications library
2:
3: #include <sys/types.h>
4: #include <sys/socket.h>
5: #ifndef __VXWORKS__
6: #include <sys/poll.h>
7: #endif
8: #include <netinet/in.h>
9: #include <arpa/inet.h>
10: #include <stdio.h>
11: #include <string.h>
12: #include <unistd.h>
13: #include <errno.h>
14: #include "sdmLib.h"
15: #include "TCPcom.h"
16:
17: #ifdef __VXWORKS__
18: #include <sockLib.h>
19: #include <selectLib.h>
20: #endif
21:
22: #ifndef WIN32
23: #define closesocket close
24: #endif
25:
26: struct sockaddr_in TCP_fsin;
27:
28: SDMLIB_API
29: int TCPpassive (int port, int maxConn)      // setup server passive port
30: { struct sockaddr_in sin;
31:   int sock;
32:   memset (&sin, 0, sizeof(sin));
33:   sin.sin_family = AF_INET;
34:   sin.sin_addr.s_addr = INADDR_ANY;
35:   sin.sin_port = htons (static_cast<u_int16_t>(port));
36:   sock = socket (PF_INET, SOCK_STREAM, 0);
37:   const int one = 1;
38: #ifdef __VXWORKS__
39:   if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (char*)&one, sizeof(int)) == -1)
```

```

40: #else
41:   if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (void*)&one, sizeof(int)) == -1)
42: #endif
43:   {
44:     perror("TCPpassive: setsockopt");
45:   }
46:   if (bind (sock, reinterpret_cast<struct sockaddr *>(&sin), sizeof(sin)) < 0)
47:   {   printf("Unable to bind the socket! (port:%d) \n",port);
48:     perror("TCPpassive: bind");
49:     closesocket(sock);
50:     return IP_SOCKET_INVALID;
51:   }
52:   if (listen(sock, maxConn) < 0) {
53:     perror("TCPPassive: listen");
54:     closesocket(sock);
55:     return IP_SOCKET_INVALID;
56:   }
57:   return sock;
58: }
59:
60: SDMLIB_API
61: int TCPaccept (int sock, struct sockaddr_in *sin)
62: {
63:   socklen_t alen = sizeof(struct sockaddr_in);
64: #ifndef __VXWORKS__
65:   int sock2 = accept (sock, reinterpret_cast<struct sockaddr *>(sin), (socklen_t*)&alen);
66: #else
67:   int sock2 = accept (sock, reinterpret_cast<struct sockaddr *>(sin), (int*)&alen);
68: #endif
69:   if (sock2 < 0) {
70:     //If the sock is invalid, it may have been shutdown prior to an accept, but the thread is live
71:     if (errno != EINVAL)
72:       perror("TCPserv_accept");
73:     return IP_SOCKET_INVALID;
74:   }
75:   return sock2;
76: }
77:
78: SDMLIB_API
79: int TCPserv_accept (int sock)
80: {

```

```

81: return TCPaccept(sock, &TCP_fsin);
82: }
83:
84: SDMLIB_API
85: int TCPserv_recv (int sock, void *buf, size_t length) // recv from anyone (passive server socket)
86: {
87:     socklen_t alen = sizeof(TCP_fsin);
88: #ifndef __VXWORKS__
89:     return recvfrom (sock, (char*)buf, length, 0,
90:         reinterpret_cast<struct sockaddr *>(&TCP_fsin), (socklen_t*)&alen);
91: #else
92:     return recvfrom (sock, (char*)buf, length, 0,
93:         reinterpret_cast<struct sockaddr *>(&TCP_fsin), (int*)&alen);
94: #endif
95: }
96:
97: SDMLIB_API
98: int TCPserv_reply (int sock, const void *buf, size_t length) // reply to last message received on server
99: {
100:     return sendto (sock, (char*)buf, length, 0,
101:         reinterpret_cast<struct sockaddr *>(&TCP_fsin), sizeof(TCP_fsin));
102: }
103:
104: SDMLIB_API
105: int TCPserv_replyto (int sock, const void* buf, size_t length, const struct sockaddr_in* s) // reply to
some previous message received on server
106: {
107:     struct sockaddr_in sin;
108:     memset(&sin,0,sizeof(sin));
109:     sin.sin_addr = s->sin_addr;
110:     sin.sin_port = static_cast<u_int16_t>(htons(s->sin_port));
111:     sin.sin_family = s->sin_family;
112:     return sendto (sock, (char*)buf, length, 0,
113:         reinterpret_cast<struct sockaddr *>(&sin), sizeof(sin));
114: }
115:
116: // send to (unconnected sock)
117: SDMLIB_API
118: int TCPsendto (const char *ipaddr, int port, const void *buf, size_t length)
119: { struct sockaddr_in sin;
120:     int sock, status;

```

```

121:  memset (&sin, 0, sizeof(sin));
122:  sin.sin_family = AF_INET;
123:  sin.sin_addr.s_addr = inet_addr ((char*)ipaddr);
124:  sin.sin_port = htons (static_cast<u_int16_t>(port));
125:  sock = socket (PF_INET, SOCK_STREAM, 0);
126:  status = sendto (sock, (char*)buf, length, 0,
127:                  reinterpret_cast<struct sockaddr *>(&sin), sizeof(sin));
128:  closesocket (sock);
129:  return status;
130: }
131:
132: SDMLIB_API
133: int TCPrecvfrom (int port, void *buf, size_t length) // recv from anyone (unconnected)
134: { struct sockaddr_in sin;
135:   int sock, status;
136:   memset (&sin, 0, sizeof(sin));
137:   sin.sin_family = AF_INET;
138:   sin.sin_addr.s_addr = INADDR_ANY;
139:   sin.sin_port = htons (static_cast<u_int16_t>(port));
140:   sock = socket (PF_INET, SOCK_STREAM, 0);
141:   if (bind (sock, reinterpret_cast<struct sockaddr *>(&sin),sizeof(sin)) < 0)
142:   { perror ("TCPrecvfrom: ");
143:     return -1;
144:   }
145:   status = recv (sock, (char*)buf, length, 0);
146:   closesocket (sock);
147:   return status;
148: }
149:
150: /* NOTE: pip must be in network byte order */
151: SDMLIB_API
152: int TCPconnect (unsigned int pip, int port) // connected TCP socket for TCPsend() &
TCPrecv()
153: { struct sockaddr_in sin; // pip is IP address in integer (long) form
154:   int sock;
155:   memset (&sin, 0, sizeof(sin));
156:   sin.sin_family = AF_INET;
157:   sin.sin_port = htons (static_cast<u_int16_t>(port));
158:   sin.sin_addr.s_addr = pip;
159:   sock = socket (PF_INET, SOCK_STREAM, 0);
160:   if(connect (sock, reinterpret_cast<struct sockaddr *>(&sin),sizeof(sin)) < 0)

```



```

161:  {
162:      struct in_addr in = {pip};
163:
164:      printf("Unable to connect to %s:%d \n",inet_ntoa(in),port);
165:      perror("TCPconnect");
166:      return IP_SOCKET_INVALID;
167:  }
168:  return sock;
169: }
170:
171: SDMLIB_API
172: int TCPconnect (const char *ipaddr, int port)    // connected TCP socket for TCPsend() & TCPrecv()
173: {
174:     int sock;
175:     in_addr_t pip = inet_addr ((char*)ipaddr);    // ipaddr is in number-dot notation
176:     sock = TCPconnect (pip, port);
177:     if(sock == IP_SOCKET_INVALID)
178:     {
179:         printf("Failed to connect to %s:%d \n",ipaddr,port);
180:     }
181:     return sock;
182: }
183:
184: SDMLIB_API
185: int TCPsend (int sock, const void *buf, size_t length) // use with connected TCP socket
186: { return send (sock, (char*)buf, length, 0);
187: }
188:
189: SDMLIB_API
190: int TCPrecv (int sock, void *buf, size_t length) // use with connected TCP socket
191: {
192:     int iReturn = 0;
193:     bool bTryAgain = false;
194:     int iAttemptCount = 0;
195:     const int MAX_ATTEMPTS = 5;
196:     do
197:     {
198:         bTryAgain = false;
199:         iReturn = recv (sock, (char*)buf, length, 0);
200:         if (iReturn == -1)
201:         {

```

```

202:         if (errno == EINTR)
203:         {
204:             bTryAgain = true;
205:             if (++iAttemptCount >= MAX_ATTEMPTS)
206:                 bTryAgain = false;
207:         }
208:     }
209: } while (bTryAgain);
210:
211: return iReturn;
212: }
213:
214: SDMLIB_API
215: void TCPgetip (struct sockaddr_in *p)
216: {
217:     p->sin_addr = TCP_fsin.sin_addr;
218:     p->sin_port = ntohs(static_cast<u_int16_t>(TCP_fsin.sin_port));
219:     p->sin_family = TCP_fsin.sin_family;
220: }
221:
222: /* NOTE: timeout is in milliseconds */
223: SDMLIB_API
224: int TCPavail (int sock, int timeout)
225: {
226: #ifndef __VXWORKS__
227:     struct pollfd ufd;
228:     ufd.fd = sock;
229:     ufd.events = POLLIN | POLLPRI;
230:     return poll(&ufd, 1U, timeout) > 0;
231: #else
232:     struct fd_set fds;
233:     FD_SET(sock, &fds);
234:     timeval timeoutVal;
235:     timeoutVal.tv_sec = timeout;
236:     return (select(sock + 1, &fds, NULL, NULL, &timeoutVal) > 0);
237: #endif
238: }
239:
240: SDMLIB_API
241: int TCPshutdown (int sock)
242: {

```

```
243: #ifndef WIN32
244:     return shutdown(sock, SHUT_RDWR);
245: #else
246:     return shutdown(sock, SD_BOTH);
247: #endif
248: }
249:
250: SDMLIB_API
251: int TCPclose (int sock)
252: {
253:     return closesocket(sock);
254: }
```

File: sdm/common/SDMComHandle.h

```
1: #ifndef _SDM_COM_HANDLE_H_
2: #define _SDM_COM_HANDLE_H_
3:
4: #include <netinet/in.h>
5: #include "UDPcom.h"
6: #include "sdmLib.h"
7:
8: class SDMLIB_API SDMComHandle
9: {
10: public:
11:     SDMComHandle();
12:     SDMComHandle(int Sock, bool TCP, const sockaddr_in* Address);
13:     SDMComHandle(const SDMComHandle& right);
14:     SDMComHandle& operator=(const SDMComHandle& right);
15:     ~SDMComHandle();
16:
17:     int GetSock () const { return m_iSock; }
18:     bool IsValidHandle() const { return m_iSock != IP_SOCKET_INVALID; }
19:     bool IsTcp() const { return m_bIsTcp; }
20:     const struct sockaddr_in* GetAddress() const { return &m_sinAddress; }
21:     int GetPort() const { return m_sinAddress.sin_port; }
22:     void Set(int Sock, bool TCP, const sockaddr_in* Address);
23:     void DoCleanup();
24: private:
25:     int m_iSock;
26:     bool m_bIsTcp;
27:     struct sockaddr_in m_sinAddress;
28: };
29:
30: static const SDMComHandle COM_HANDLE_INVALID;
31:
32: #endif
```

File: sdm/common/message_defs.h

```
1: // SDM message definitions
2: //   Rev 0.1   Cannon, 1/05
3: //   Rev 0.2   Cannon, 1/19
4: //   Rev 0.71  Sundberg, 13APR2005
5: #ifndef __MESSAGE_DEFS_H_
6: #define __MESSAGE_DEFS_H_
7:
8: //Message types
9: #define SDM_RegPM      'a' /*formerly SDM_ReqTask*/
10: #define SDM_Task      'b'
11: #define SDM_ReqCode   'c'
12: #define SDM_Code     'd'
13: #define SDM_Kill      'e' /*CMDKillProc*/
14: #define SDM_Preempt   'f'
15: #define SDM_ProcStatus 'g'
16: #define SDM_Heartbeat 'h'
17: #define SDM_PostTask  'i' /*CMDPostTask*/
18: #define SDM_ChgPriority 'j'
19: #define SDM_ChgMode   'k'
20: #define SDM_Consume   'l' /*CMDSubscribe*/
21: #define SDM_Cancel    'm' /*CMDCancelSub*/
22: #define SDM_ReqReg    'n' /*CMDReqReg*/
23: #define SDM_RegInfo   'o' /*CMDRegInfo*/
24: #define SDM_SubError  'p'
25: #define SDM_xTEDS     'q' /*CMDxTEDS*/
26: #define SDM_CancelxTEDS 'r' /*CMDCancelxTEDS*/
27: #define SDM_SensorCtrl 's'
28: #define SDM_Data      't' /*CMDPublish*/
29: #define SDM_Subreqst  'u' /*CMDSubreqst*/
30: #define SDM_Deletesub 'v' /*CMDDeletesub*/
31: #define SDM_ReqMode   'w'
32: #define SDM_Service   'x' /*no equivalent*/
33: #define SDM_Serreqst  'y' /*no equivalent*/
34: #define SDM_Command   'z' /*no equivalent*/
35: #define SDM_ReqxTEDS  'A' /*no equivalent*/
36: #define SDM_xTEDSInfo 'B' /*no equivalent*/
37: #define SDM_Error     'C' /*no equivalent*/
38: #define SDM_Ready     'D' /*no equivalent*/
39: #define SDM_TaskFinished 'E' /*no equivalent*/
```

```

40: #define SDM_ReqPort      'F' /*no equivalent*/
41: #define SDM_ACK          'G' /*no equivalent*/
42: #define SDM_Tat          'H' /*no equivalent*/
43: #define SDM_Search       'I' /*no equivalent*/
44: #define SDM_SearchReply   'J' /*no equivalent*/
45: #define SDM_DMLeader      'K' /*no equivalent*/
46: #define SDM_Election      'L' /*no equivalent*/
47: #define SDM_xTEDSUpdate   'M' /*no equivalent*/
48: #define SDM_VarReq        'N' /*no equivalent*/
49: #define SDM_VarInfo       'O' /*no equivalent*/
50: #define SDM_Hello         'P'
51: #define SDM_Register      'Q'
52: #define SDM_ID            'R'
53: #define SDM_WritexTEDS    'W' /*no equivalent*/
54: #define SDM_TaskError      'T'
55:
56: //Error codes
57: #define SDM_ERROR_CODE_NOT_FOUND 0x01
58: #define SDM_ERROR_SUBSCRIPTION_REJECTED 0x02
59: #define SDM_ERROR_INVALID_XTEDS 0x03
60: #define SDM_ERROR_COMMAND_REJECTED 0x04
61:
62: //TM Mode codes
63: #define MODE_SOFT_RESET      1
64: #define MODE_HARD_RESET     2
65:
66: //Ports for managers
67: #define PORT_TM      3510      /*port for Task Manager*/
68: #define PORT_TM_MONITOR 3511    /*port for Task Manager monitor process*/
69: #define PORT_DM      3504      /*port for Data Manager*/
70: #define PORT_DM_ELECTION 3505    /*port for Data Manager elections to take place*/
71: #define PORT_DM_TEMP 3509      /*temporary port used by the DM for IPC*/
72: #define PORT_DM_MONITOR 3513    /*port for Data Manager monitor process*/
73: #define PORT_SM      3506      /*port for Sensor Manager*/
74: #define PORT_SM_MONITOR 3507    /*port for Sensor Manager monitor process*/
75: #define PORT_PM      3508      /*port for Process Manager*/
76: #define PORT_PM_MONITOR 3512    /*port for Process Manager monitor process*/
77: #define PORT_SPA1_MANAGER 3514 /*port for SPA-1 Manager*/
78: #define PORT_APP_START 4000     /*the first port assigned to applications*/
79: #define PORT_APP_MAX 65535     /*the max port number that can be assigned to applications*/
80:

```

```

81: // BUFSIZE is the largest size for a UDP datagram
82: // LARGE_MSG_BUFSIZE should be used any time SDMxTEDS is to be received in a message
buffer
83: #define BUFSIZE      24288      /*maximum size of a SDM message*/
84: #if BUFSIZE < 24288
85: // If BUFSIZE is relatively small, transmit large xTEDS via TCP, otherwise
86: // it is assumed that an xTEDS can fit in a datagram
87: # define TCP_TRANSMIT_OF_LARGE_XTEDS  1
88: # define LARGE_MSG_BUFSIZE  3*8096      // Upper bound on SDMxTEDS message size
89: #else // BUFSIZE >= 24288
90: # define LARGE_MSG_BUFSIZE  BUFSIZE
91: #endif // #if BUFSIZE < 24288
92:
93: #define XTEDS_MAX_ITEM_NAME_SIZE  33
94: #define MSG_DEF_SIZE  (BUFSIZE-27)/2      /*Max size for a message def*/
95:
96: //Number of retries for SDMxTEDS and SDMCancelxTEDS
97: #ifdef __uCLinux__
98: #define NUMRETRIES 99999
99: #else
100: #define NUMRETRIES 5
101: #endif
102:
103: //Minimum time for heartbeat messages to be sent in seconds
104: #define HEARTBEAT_INTERVAL 5
105:
106: #define MAXSTRLEN  8
107: #define MAX_FILENAME_SIZE 128
108: #define MAX_TCP_CONNECTIONS 10
109:
110: //optional compile-time flags
111:
112: #endif

```

File: sdm/common/ErrorUtils.h

```
1: #ifndef _SDM_ERROR_H_
2: #include "sdmLib.h"
3:
4: // A central location for functions dealing with SDM related errors.
5:
6: class SDMLIB_API ErrorUtils
7: {
8: public:
9:   static void MemoryAllocError(const char* ErrorString);
10:  static void MemoryAllocError();
11: };
12:
13: #endif
```


File: sdm/common/asim.h

```
1: #ifndef __ASIM_H_
2: #define __ASIM_H_
3:
4: #define    ASIM_ANALOG_SOURCE    0x0001
5: #define    ASIM_THERMOCOUPLE    0x0002
6: #define    ASIM_VGI_AMP    0x0003
7: #define    ASIM_DIGITAL_IO    0x0004
8:
9: #define    ASIM_PRODUCT_ID    0xC001
10: #define    ASIM_TIMEOUT_SEC    0xC002
11: #define    ASIM_ROBUSTHUB_CMD    0xC003
12: #define    ASIM_PATH    0xC004
13:
14: #define    VGI_AMP_STAGE_TWO_GAIN    0x01
15: #define    VGI_AMP_STAGE_ONE_GAIN    0x02
16: #define    VGI_AMP_OFFSET    0x03
17:
18: #define    ANALOG_SOURCE_DAC_OUT    0x01
19:
20: #define    DIGITAL_IO_PROGRAM    0x01
21: #define    DIGITAL_IO_IO    0x02
22:
23: #define    ASIM_WRITE_FLASH    0xFA
24: #define    ASIM_ERASE_FLASH    0xFB
25: #define    ASIM_READW_FLASH    0xFC
26: #define    ASIM_READB_FLASH    0xFD
27:
28: #endif
```

File: sdm/common/message/SDMTaskError.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMTaskError.h"
4:
5: SDMTaskError::SDMTaskError():source(),status(0), pid(0)
6: {
7:     MsgName = SDM_TaskError;
8:     filename[0] = '\0';
9:     total_length = 19;
10: }
11:
12: long SDMTaskError::Send()
13: {
14:     return SendDM();
15: }
16:
17: long SDMTaskError::Marshal(char* buf)
18: {
19:     int cur;
20:     cur = HEADER_SIZE;
21:     cur += source.Marshal(buf,cur);
22:     PUT_INT(&buf[cur], status);
23:     cur += sizeof(status);
24:     PUT_UINT(&buf[cur], pid);
25:     cur += sizeof(pid);
26:     memcpy (&buf[cur], &filename, strlen(filename));
27:     cur += strlen(filename);
28:     buf[cur] = '\0';
29:     cur++;
30:     msg_length = cur - HEADER_SIZE;
31:     MarshalHeader(buf);
32:     return cur;
33: }
34:
35: long SDMTaskError::Unmarshal(const char* buf)
36: {
37:     int cur;
38:     cur = UnmarshalHeader(buf);
39:     if(cur == SDM_INVALID_MESSAGE)
```

```

40: {
41:     return SDM_INVALID_MESSAGE;
42: }
43: if(total_length>msg_length)
44: {
45:     return SDM_INVALID_MESSAGE;
46: }
47: cur += source.Unmarshal(buf,cur);
48: status = GET_INT(&buf[cur]);
49: cur += sizeof (status);
50: pid = GET_UINT(&buf[cur]);
51: cur += sizeof (pid);
52:
53: unsigned int uiCurLength = strlen(buf + cur);
54: strncpy(filename, buf + cur, sizeof(filename));
55: if (uiCurLength > sizeof(filename) - 1)
56:     filename[sizeof(filename) - 1] = '\0';
57: cur += uiCurLength + 1;
58: #ifdef BUILD_WITH_MESSAGE_LOGGING
59: Logger.MessageReceived(*this);
60: #endif
61: return cur;
62: }
63: int SDMTaskError::MsgToString(char *str_buf, int length) const
64: {
65:     char source_id[40];
66:     if (length < 8096)
67:         return 0;
68:     source.IDToString(source_id,40);
69: #ifdef WIN32
70:     sprintf(str_buf,
71: #else
72:     snprintf(str_buf,length,
73: #endif
74:         "SDMTaskError %ld:%ld %ld bytes, Source: %s PID: %u Filename: %s Result: %d", sec,
75:         subsec, msg_length, source_id, pid, filename, status);
76:     return strlen(str_buf);
77: }

```

File: sdm/common/message/SDMMessage_ID.h

```
1: #ifndef __SDMMESSAGE_ID_H_
2: #define __SDMMESSAGE_ID_H_
3:
4: #include "../sdmLib.h"
5:
6: class SDMLIB_API SDMMessage_ID
7: {
8: public:
9:  SDMMessage_ID();
10:  SDMMessage_ID(unsigned char interface_id, unsigned char msg_id);
11:  SDMMessage_ID(int interface_id, int msg_id);
12:  void setInterface(unsigned char interface_num) { _interface = interface_num; } //Sets the interface
id number for this Message_ID
13:  void setMessage(unsigned char msg_num) { _message = msg_num; } //Sets the message
id number for this Message_ID
14:  void setInterface(int interface_num);
15:  void setMessage(int msg_num);
16:  unsigned char getInterface() const { return _interface; } //Gets the interface id number for this
Message_ID
17:  unsigned char getMessage() const { return _message; }; //Gets the message id number for this
Message_ID
18:  short getInterfaceMessagePair() const;
19:  bool operator== (const SDMMessage_ID &other) const;
20:  bool operator== (const short &other) const;
21:  SDMMessage_ID& operator= (const SDMMessage_ID &other);
22:  SDMMessage_ID& operator= (const int value);
23:  int Marshal(char *buf, int start);
24:  int Unmarshal(const char *buf, int start);
25:  int IDToString(char *buf, int length) const;
26: private:
27:  unsigned char _interface;
28:  unsigned char _message;
29: };
30:
31: #endif
```

File: sdm/common/message/SDMSearch.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMSearch.h"
4:
5: SDMSearch::SDMSearch():source(),destination(),reply(SDM_SEARCH_CURRENT),id(0)
6: {
7:     MsgName = SDM_Search;
8:     reg_expr[0] = '\0';
9:     total_length = 24;
10: }
11:
12: long SDMSearch::Send()
13: {
14:     return SendDM();
15: }
16:
17: long SDMSearch::Recv(long port)
18: {
19:     int result;
20:     result = RecvFrom(port);
21:     if (result <= 0)
22:         return SDM_MESSAGE_RECV_ERROR;
23:     return result;
24: }
25:
26: long SDMSearch::Marshal(char* buf)
27: {
28:     int cur;
29:     long reg_expr_msg_length;
30:
31:     reg_expr_msg_length = (long)strlen(reg_expr);
32:     cur = HEADER_SIZE;
33:     cur += source.Marshal(buf,cur);
34:     cur += destination.Marshal(buf,cur);
35:     PUT_UCHAR (&buf[cur], reply);
36:     cur += sizeof(reply);
37:     PUT_USHORT (&buf[cur], id);
38:     cur += sizeof(id);
39:     memcpy(buf+cur,reg_expr,reg_expr_msg_length);
```

```

40: cur += reg_expr_msg_length;
41: buf[cur] = '\0';
42: cur++;
43: msg_length = cur - HEADER_SIZE;
44: MarshalHeader(buf);
45: return cur;
46: }
47:
48: long SDMSearch::Unmarshal(const char* buf)
49: {
50: int cur;
51: unsigned int reg_expr_msg_length;
52:
53: //unmarshal header
54: cur = UnmarshalHeader(buf);
55: if(cur == SDM_INVALID_MESSAGE)
56: {
57:     return SDM_INVALID_MESSAGE;
58: }
59: if(total_length>msg_length)
60: {
61:     return SDM_INVALID_MESSAGE;
62: }
63: cur += source.Unmarshal(buf,cur);
64: cur += destination.Unmarshal(buf,cur);
65: reply = GET_UCHAR (&buf[cur]);
66: cur += sizeof(reply);
67: id = GET_USHORT (&buf[cur]);
68: cur += sizeof(id);
69: //
70: // Get the regular expression
71: reg_expr_msg_length = (unsigned int)strlen(buf+cur);
72: strncpy(reg_expr, buf + cur, sizeof(reg_expr));
73: if (reg_expr_msg_length > sizeof(reg_expr) - 1)
74:     reg_expr[sizeof(reg_expr) - 1] = '\0';
75: cur += reg_expr_msg_length+1;
76:
77: #ifdef BUILD_WITH_MESSAGE_LOGGING
78: Logger.MessageReceived(*this);
79: #endif
80: return cur;

```

```

81: }
82:
83: int SDMSearch::MsgToString(char *str_buf, int length) const
84: {
85:     char source_id[40];
86:     char destination_id[40];
87:
88:     if (length < 8096)
89:         return 0;
90:     source.IDToString(source_id, 40);
91:     destination.IDToString(destination_id, 40);
92: #ifdef WIN32
93:     sprintf(str_buf,
94: #else
95:     snprintf(str_buf,length,
96: #endif
97:         "SDMSearch %ld:%ld %ld bytes, Source: %s Destination: %s Reply: %d ID: %d RegExpr: %s",
98:         sec, subsec, msg_length, source_id, destination_id, reply, id, reg_expr);
99:     return (int)strlen(str_buf);
100: }

```

File: sdm/common/message/SDMElection.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "SDMElection.h"
4:
5: SDMElection::SDMElection()
6: {
7:     MsgName = SDM_Election;
8:     total_length = 0;
9: }
10: SDMElection::~SDMElection()
11: {
12: }
13: long SDMElection::Marshal(char *buf)
14: {
15:     int cur = HEADER_SIZE;
16:
17:     MarshalHeader(buf);
18:     return cur;
19: }
20:
21: long SDMElection::Unmarshal(const char*buf)
22: {
23:     int cur = 0;
24:     cur = UnmarshalHeader(buf);
25:     if (cur == SDM_INVALID_MESSAGE)
26:         return SDM_INVALID_MESSAGE;
27: #ifdef BUILD_WITH_MESSAGE_LOGGING
28:     Logger.MessageReceived(*this);
29: #endif
30:     return cur;
31: }
32:
33: //SDMElection messages send via Broadcast
34: long SDMElection::Send()
35: {
36:     return SendBroadcast();
37: }
38:
39: int SDMElection::MsgToString(char *str_buf, int length) const
```



```
40: {
41: if (length < 8096)
42:     return 0;
43: #ifdef WIN32
44: sprintf(str_buf,
45: #else
46: snprintf(str_buf,length,
47: #endif
48:     "SDMElection %ld:%ld %ld bytes", sec, subsec, msg_length);
49: return (int)strlen(str_buf);
50: }
```

File: sdm/common/message/SDMID.h

```
1: #ifndef __SDM_ID_H_
2: #define __SDM_ID_H_
3:
4: #include "SDMmessage.h"
5:
6:
7: class SDMLIB_API SDMID: public SDMmessage
8: {
9: public:
10:  SDMComponent_ID destination;           //The destination of the ID msg
11:
12:  SDMID();
13:  long Marshal(char *buf);
14:  long Unmarshal(const char * buf);
15:  int MsgToString(char *str_buf, int length) const;
16: };
17: #endif
```

File: sdm/common/message/SDMDeletesub.h

```
1: #ifndef __SDM_DELETESUB_H_
2: #define __SDM_DELETESUB_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: class SDMLIB_API SDMDeletesub: public SDMmessage
8: {
9: public :
10: SDMComponent_ID source;      //data provider
11: SDMComponent_ID destination; //data consumer
12: SDMMessage_ID msg_id;
13: SDMDeletesub();
14: long Recv(long port); SDM_DEPRECATED
15: long Send();
16: long Marshal(char* buf);
17: long Unmarshal(const char* buf);
18: int MsgToString(char *str_buf, int length) const;
19: };
20:
21: #endif
```

File: sdm/common/message/SDMConsume.cpp

```
1: #include "SDMConsume.h"
2: #include <stdio.h>
3: #include <string.h>
4:
5: SDMConsume::SDMConsume():source(),destination(),msg_id(0,0)
6: {
7:     MsgName = SDM_Consume;
8:     total_length = 22;
9: }
10:
11: long SDMConsume::Send()
12: {
13:     return SendDM();
14: }
15:
16: long SDMConsume::Marshal(char* buf)
17: {
18:     int cur;
19:     cur = HEADER_SIZE;
20:     cur += source.Marshal(buf,cur);
21:     cur += destination.Marshal(buf,cur);
22:     cur += msg_id.Marshal(buf, cur);
23:     msg_length = cur - HEADER_SIZE;
24:     MarshalHeader(buf);
25:     return cur;
26: }
27:
28: long SDMConsume::Unmarshal(const char* buf)
29: {
30:     int cur;
31:     cur = UnmarshalHeader(buf);
32:     if(cur==SDM_INVALID_MESSAGE)
33:     {
34:         return SDM_INVALID_MESSAGE;
35:     }
36:     if(total_length!=msg_length)
37:     {
38:         return SDM_INVALID_MESSAGE;
39:     }
```

```

40: cur += source.Unmarshal(buf,cur);
41: cur += destination.Unmarshal(buf,cur);
42: cur += msg_id.Unmarshal(buf, cur);
43: #ifdef BUILD_WITH_MESSAGE_LOGGING
44: Logger.MessageReceived(*this);
45: #endif
46: return HEADER_SIZE+msg_length;
47: }
48: int SDMConsume::MsgToString(char *str_buf, int length) const
49: {
50: char source_id[40];
51: char dest_id[40];
52: char message_id[30];
53:
54: if (length < 8096)
55:     return 0;
56: source.IDToString(source_id, 40);
57: destination.IDToString(dest_id, 40);
58: msg_id.IDToString(message_id, 30);
59: #ifdef WIN32
60: sprintf(str_buf,
61: #else
62: snprintf(str_buf,length,
63: #endif
64:         "SDMConsume %ld:%ld %ld bytes, Source: %s Dest: %s %s" ,sec, subsec, msg_length,
source_id, dest_id, message_id);
65: return (int)strlen(str_buf);
66: }

```

File: sdm/common/message/SDMCode.h

```
1: #ifndef __SDM_CODE_H_
2: #define __SDM_CODE_H_
3:
4: #include "SDMmessage.h"
5:
6: //This class is intended only for use by SDM core components
7:
8: class SDMLIB_API SDMCode : public SDMmessage
9: {
10: public:
11: unsigned short seq_num;          //A sequence number to reorder packets in
12: unsigned short num_segments; //The total number of packets the code is split into
13: char filename[MAX_FILENAME_SIZE]; //The name of the code
14: unsigned short code_length; //The length of the code
15: int c_sum; //The checksum for the code
16: char code[BUFSIZE - MAX_FILENAME_SIZE - 21]; //The buffer for the piece of code
17: SDMCode();
18: long Marshal(char* buf);
19: long Unmarshal(const char* buf);
20: bool isCorrupt();
21: int MsgToString(char *str_buf, int length) const;
22: };
23:
24:
25:
26: #endif
```

File: sdm/common/message/SDMService.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMService.h"
4:
5: SDMService::SDMService():source(),destination(),command_id(0,0),length(0),seq_num(0)
6: {
7:     MsgName = SDM_Service;
8:     data[0] = '\0';
9:     total_length = 24;
10: }
11:
12: long SDMService::Send()
13: {
14:     return SendDM();
15: }
16:
17: long SDMService::Recv(long port)
18: {
19:     int result;
20:     result = RecvFrom(port);
21:     if (result <= 0)
22:         return SDM_MESSAGE_RECV_ERROR;
23:     return result;
24: }
25:
26: long SDMService::Marshal(char* buf)
27: {
28:     int cur;
29:     cur = HEADER_SIZE;
30:     cur += source.Marshal(buf,cur);
31:     cur += destination.Marshal(buf,cur);
32:     cur += command_id.Marshal(buf, cur);
33:     PUT_SHORT(&buf[cur],seq_num);
34:     cur += sizeof(seq_num);
35:     memcpy(buf+cur,&data,length);
36:     cur += length;
37:     msg_length = cur-HEADER_SIZE;
38:     MarshalHeader(buf);
39:     return cur;
```

```

40: }
41:
42: long SDMService::Unmarshal(const char* buf)
43: {
44: int cur;
45: cur = UnmarshalHeader(buf);
46: if(cur==SDM_INVALID_MESSAGE)
47: {
48:     return SDM_INVALID_MESSAGE;
49: }
50: if(total_length>msg_length)
51: {
52:     return SDM_INVALID_MESSAGE;
53: }
54: cur += source.Unmarshal(buf,cur);
55: cur += destination.Unmarshal(buf,cur);
56: cur += command_id.Unmarshal(buf, cur);
57: seq_num = GET_SHORT(&buf[cur]);
58: cur += sizeof(seq_num);
59: length = msg_length + HEADER_SIZE - cur;
60: memcpy(&data,buf+cur,length);
61: #ifdef BUILD_WITH_MESSAGE_LOGGING
62: Logger.MessageReceived(*this);
63: #endif
64: return msg_length+HEADER_SIZE;
65: }
66: int SDMService::MsgToString(char *str_buf, int length) const
67: {
68: char source_id[40];
69: char dest_id[40];
70: char data_section[511];
71: char cmd_id[30];
72: int i, j;
73:
74: if (length < 8096)
75:     return 0;
76: for (i = 0, j = 0; j < this->length && i < 510; i+=2, j++)
77: {
78:     sprintf(&data_section[i], "%.2x", data[j]);
79: }
80: data_section[i] = '\0';

```



```
81:
82: source.IDToString(source_id, 40);
83: destination.IDToString(dest_id, 40);
84: command_id.IDToString(cmd_id, 30);
85: #ifdef WIN32
86: sprintf(str_buf,
87: #else
88: snprintf(str_buf,length,
89: #endif
90:      "SDMService %ld:%ld %ld bytes, Source: %s Dest: %s CommandID: %s seq_num: %d [DATA
SECTION: %s ]", sec, subsec, msg_length, source_id, dest_id, cmd_id, seq_num, data_section);
91: return (int)strlen(str_buf);
92: }
```

File: sdm/common/message/SDMAck.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "SDMAck.h"
4:
5: SDMAck::SDMAck() : error(0)
6: {
7:     MsgName = SDM_ACK;
8:     total_length = 2;
9: }
10:
11: long SDMAck::Send()
12: {
13:     return SendDM();
14: }
15:
16: long SDMAck::Send(const SDMComHandle& Handle)
17: {
18:     if (Handle.IsValidHandle())
19:         return SendReplyTo(Handle.GetSock(), Handle.GetAddress(), Handle.IsTcp());
20:     return -1;
21: }
22:
23: long SDMAck::Marshal(char *buf)
24: {
25:     int cur = HEADER_SIZE;
26:     PUT_SHORT(buf+cur,error);
27:     cur += sizeof(error);
28:     msg_length = cur-HEADER_SIZE;
29:     MarshalHeader(buf);
30:     return cur;
31: }
32:
33: long SDMAck::Unmarshal(const char * buf)
34: {
35:     int cur = UnmarshalHeader(buf);
36:     if (cur == SDM_INVALID_MESSAGE || total_length != msg_length)
37:     {
38:         return SDM_INVALID_MESSAGE;
39:     }
```

```

40: error = GET_SHORT(buf+cur);
41: cur += sizeof(error);
42: #ifdef BUILD_WITH_MESSAGE_LOGGING
43: Logger.MessageReceived(*this);
44: #endif
45: return cur;
46: }
47:
48: int SDMAck::MsgToString(char *buf, int length) const
49: {
50: if (length < 8096)
51:     return 0;
52: #ifdef WIN32 // cleanup - remove duplicate code for maintainability
53: sprintf(buf,
54: #else
55:     snprintf(buf,length,
56: #endif
57:     "SDMAck %ld:%ld %ld bytes, error: %hd", sec, subsec, msg_length, error);
58: return (int)strlen(buf);
59: }

```

File: sdm/common/message/SDMxTEDSInfo.h

```
1: #ifndef __SDM_XTEDSINFO_H_
2: #define __SDM_XTEDSINFO_H_
3:
4: #include "SDMmessage.h"
5:
6: class SDMLIB_API SDMxTEDSInfo: public SDMmessage
7: {
8: public :
9:   SDMComponent_ID source;    //unique id of xTEDS provider
10:  char xTEDS[LARGE_MSG_BUFSIZE - 21];    //xTEDS registered with SDM system
11:  SDMxTEDSInfo();
12:  long Send();
13:  long Marshal(char* buf);
14:  long Unmarshal(const char* buf);
15:  long Recv(long port); SDM_DEPRECATED
16:  long Send(const SDMComponent_ID& destination);
17:  long MarshalEmpty(char* buf);
18:  long SendEmpty(const SDMComponent_ID& destination);
19:  int MsgToString(char *str_buf, int length) const;
20:
21: private:
22:  bool emptyflag;    //A flag to be set when wanting to send an empty message
23: };
24:
25: #endif
```

File: sdm/common/message/SDMPostTask.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMPostTask.h"
4:
5: SDMPostTask::SDMPostTask():resources(0),priority(0),sched_interval(0),mode(0),version(0)
6: {
7:     filename[0] = '\0';
8:     MsgName = SDM_PostTask;
9:     total_length = 16;
10: }
11:
12: long SDMPostTask::Send()
13: {
14:     return SendTM();
15: }
16:
17: long SDMPostTask::Marshal(char* buf)
18: {
19:     int cur = HEADER_SIZE;
20:     PUT_USHORT (&buf[cur], resources);
21:     cur += sizeof(resources);
22:     PUT_CHAR (&buf[cur], priority);
23:     cur += sizeof(priority);
24:     PUT_INT (&buf[cur], sched_interval);
25:     cur += sizeof(sched_interval);
26:     PUT_INT (&buf[cur], mode);
27:     cur += sizeof(mode);
28:     PUT_INT (&buf[cur], version);
29:     cur += sizeof(version);
30:     msg_length = (short)strlen(filename);
31:     memcpy(buf+cur,&filename,msg_length);
32:     cur += msg_length;
33:     buf[cur] = '\0';//null termination
34:     cur++;
35:     msg_length = cur - HEADER_SIZE;
36:     MarshalHeader(buf);
37:     return msg_length + HEADER_SIZE ;
38: }
39:
```

```

40: long SDMPostTask::Unmarshal(const char* buf)
41: {
42: int cur;
43: cur = UnmarshalHeader(buf);
44: if(cur==SDM_INVALID_MESSAGE)
45: {
46:     return SDM_INVALID_MESSAGE;
47: }
48: if(total_length>msg_length)
49: {
50:     return SDM_INVALID_MESSAGE;
51: }
52: resources = GET_USHORT(&buf[cur]);
53: cur += sizeof(resources);
54: priority = GET_CHAR (&buf[cur]);
55: cur += sizeof(priority);
56: sched_interval = GET_INT(&buf[cur]);
57: cur += sizeof(sched_interval);
58: mode = GET_INT(&buf[cur]);
59: cur += sizeof(mode);
60: version = GET_INT(&buf[cur]);
61: cur += sizeof(version);
62: //
63: // Get the filename
64: unsigned int uiCurLength = (unsigned int)strlen(buf + cur);
65: strncpy(filename, buf + cur, sizeof(filename));
66: if (uiCurLength > sizeof(filename) - 1)
67:     filename[sizeof(filename) - 1] = '\0';
68:
69: #ifdef BUILD_WITH_MESSAGE_LOGGING
70: Logger.MessageReceived(*this);
71: #endif
72: return msg_length+HEADER_SIZE;
73: }
74: int SDMPostTask::MsgToString(char *str_buf, int length) const
75: {
76: if (length < 8096)
77:     return 0;
78: #ifdef WIN32
79: sprintf(str_buf,
80: #else

```

```
81: snprintf(str_buf,length,
82: #endif
83:      "SDMPostTask %ld:%ld %ld bytes, Resources: %hu Priority: %d Filename: %s SchedInterval:
%d Mode: %d Version: %d", sec, subsec, msg_length, resources, priority, filename, sched_interval,
mode, version);
84: return (int)strlen(str_buf);
85: }
```

File: sdm/common/message/SDMTaskFinished.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMTaskFinished.h"
4:
5: SDMTaskFinished::SDMTaskFinished():source(),result(0), pid(0)
6: {
7:     MsgName = SDM_TaskFinished;
8:     filename[0] = '\0';
9:     total_length = 19;
10: }
11:
12: long SDMTaskFinished::Send()
13: {
14:     return SendTM();
15: }
16:
17: long SDMTaskFinished::Marshal(char* buf)
18: {
19:     int cur;
20:     cur = HEADER_SIZE;
21:     cur += source.Marshal(buf,cur);
22:     PUT_INT(&buf[cur], result);
23:     cur += sizeof(result);
24:     PUT_INT(&buf[cur], pid);
25:     cur += sizeof(pid);
26:     memcpy (&buf[cur], &filename, strlen(filename));
27:     cur += strlen(filename);
28:     buf[cur] = '\0';
29:     cur++;
30:     msg_length = cur - HEADER_SIZE;
31:     MarshalHeader(buf);
32:     return cur;
33: }
34:
35: long SDMTaskFinished::Unmarshal(const char* buf)
36: {
37:     int cur;
38:     cur = UnmarshalHeader(buf);
39:     if(cur == SDM_INVALID_MESSAGE)
```



```

40: {
41:     return SDM_INVALID_MESSAGE;
42: }
43: if(total_length>msg_length)
44: {
45:     return SDM_INVALID_MESSAGE;
46: }
47: cur += source.Unmarshal(buf,cur);
48: result = GET_INT(&buf[cur]);
49: cur += sizeof (result);
50: pid = GET_INT(&buf[cur]);
51: cur += sizeof (pid);
52:
53: unsigned int uiCurLength = strlen(buf + cur);
54: strncpy(filename, buf + cur, sizeof(filename));
55: if (uiCurLength > sizeof(filename) - 1)
56:     filename[sizeof(filename) - 1] = '\0';
57: cur += uiCurLength + 1;
58: #ifdef BUILD_WITH_MESSAGE_LOGGING
59: Logger.MessageReceived(*this);
60: #endif
61: return cur;
62: }
63: int SDMTaskFinished::MsgToString(char *str_buf, int length) const
64: {
65: char source_id[40];
66: if (length < 8096)
67:     return 0;
68: source.IDToString(source_id,40);
69: #ifdef WIN32
70: sprintf(str_buf,
71: #else
72: snprintf(str_buf,length,
73: #endif
74:     "SDMTaskFinished %ld:%ld %ld bytes, Source: %s PID: %d Filename: %s Result: %d", sec,
subsec, msg_length, source_id, pid, filename, result);
75: return strlen(str_buf);
76: }

```

File: sdm/common/message/SDMData.cpp

```
1: #include <string.h>
2: #include <unistd.h>
3: #include <stdio.h>
4: #include "SDMData.h"
5: #include "../UDPcom.h"
6:
7: SDMData::SDMData():source(),msg_id(0,0),length(0),seq_num(0),pid(PID)
8: {
9:     MsgName = SDM_Data;
10:    msg[0] = '\0';
11:    total_length = 18;
12: }
13:
14: long SDMData::Send(const SDMComponent_ID& destination,long data_length)
15: {
16:    long result;
17:    length = data_length;
18:    result = SendTo(destination);
19:    if (result <= 0)
20:        return SDM_MESSAGE_SEND_ERROR;
21:    return result;
22: }
23:
24: long SDMData::Send(const SDMComponent_ID& destination)
25: {
26:    long result;
27:    result = SendTo(destination);
28:    if (result <= 0)
29:        return SDM_MESSAGE_SEND_ERROR;
30:    return result;
31: }
32:
33: long SDMData::Recv(long port,long length)
34: {
35:    char buf[BUFSIZE];
36:    long result;
37:    if(bound == IP_SOCK_INVALID)
38:    {
39:        bound = UDPpassive(port);
```

```

40: }
41: if (bound > 0)
42: {
43:     result = UDPServ_recv(bound,buf,BUFSIZE);
44:     if (result > 0)
45:     {
46:         return Unmarshal(buf,length);
47:     }
48: }
49: return SDM_MESSAGE_RECV_ERROR;
50: }
51:
52: long SDMDData::Marshal(char* buf,long data_length)
53: {
54: int cur;
55: cur = HEADER_SIZE;
56: cur += source.Marshal(buf,cur);
57: cur += msg_id.Marshal(buf, cur);
58: PUT_LONG(&buf[cur],pid);
59: cur += sizeof(pid);
60: PUT_SHORT(&buf[cur],seq_num);
61: cur += sizeof(seq_num);
62: memcpy(buf+cur,msg,data_length);
63: cur += data_length;
64: msg_length = cur - HEADER_SIZE;
65: MarshalHeader(buf);
66: return cur;
67: }
68:
69: long SDMDData::Unmarshal(const char* buf,long length)
70: {
71: return Unmarshal(buf);
72: }
73:
74: long SDMDData::Marshal(char* buf)
75: {
76: return Marshal(buf,length);
77: }
78:
79: long SDMDData::Unmarshal(const char* buf)
80: {

```

```

81: int cur;
82: cur = UnmarshalHeader(buf);
83: if(cur==SDM_INVALID_MESSAGE)
84: {
85:     return SDM_INVALID_MESSAGE;
86: }
87: if(total_length>msg_length)
88: {
89:     return SDM_INVALID_MESSAGE;
90: }
91: cur += source.Unmarshal(buf,cur);
92: cur += msg_id.Unmarshal(buf, cur);
93: pid = GET_LONG (&buf[cur]);
94: cur += sizeof(pid);
95: seq_num = GET_SHORT(&buf[cur]);
96: cur += sizeof(seq_num);
97: memcpy(msg,buf+cur,msg_length - cur + HEADER_SIZE);
98: length = msg_length - cur + HEADER_SIZE;
99: #ifdef BUILD_WITH_MESSAGE_LOGGING
100:     Logger.MessageReceived(*this);
101: #endif
102:     return msg_length + HEADER_SIZE;
103: }
104:
105: /*short SDMDData::getInt(long start_byte) const
106: {
107:     return GET_INT (&msg[start_byte]);
108: }
109:
110: long SDMDData::getLong(long start_byte) const
111: {
112:     return GET_LONG (&msg[start_byte]);
113: }
114:
115: signed char SDMDData::getChar(long start_byte) const
116: {
117:     return GET_CHAR (&msg[start_byte]);
118: }
119:
120: unsigned char SDMDData::getByte(long start_byte) const
121: {

```

```

122:     return GET_UCHAR (&msg[start_byte]);
123: }
124:
125: float SDMData::getFloat(long start_byte) const
126: {
127:     return GET_FLOAT (&msg[start_byte]);
128: }
129:
130: double SDMData::getDouble(long start_byte) const
131: {
132:     return GET_DOUBLE (&msg[start_byte]);
133: }
134:
135: char* SDMData::getString(long start_byte) const
136: {
137:     return strdup(msg+start_byte,BUFSIZE-HEADER_SIZE-start_byte);
138: }
139:
140: void SDMData::setSocket(int sock)
141: {
142:     bound = sock;
143: }*/
144:
145: int SDMData::MsgToString(char *str_buf, int length) const
146: {
147:     char source_id[40];
148:     char data_section[511];
149:     char message_id[30];
150:     int i, j;
151:
152:     if (length < 8096)
153:         return 0;
154:     for (i = 0, j = 0; j < this->length && i < 510; i+=2, j++)
155:     {
156:         sprintf(&data_section[i], "%.2x", msg[j]);
157:     }
158:     data_section[i] = '\0';
159:
160:     msg_id.IDToString(message_id, 30);
161:     source.IDToString(source_id, 40);
162: #ifdef WIN32

```

```
163:    sprintf(str_buf,"SDMData %ld:%ld %d bytes, Source: %s %s pid: %ld seq_num: %d [DATA
SECTION: %s ] DataLength: %d", sec, subsec, msg_length, source_id, message_id, pid, seq_num,
data_section, this->length);
164: #else
165:    snprintf(str_buf,length,"SDMData %ld:%ld %d bytes, Source: %s %s pid: %ld seq_num: %d
[DATA SECTION: %s ] DataLength: %d", sec, subsec, msg_length, source_id, message_id, pid,
seq_num, data_section, this->length);
166: #endif
167:    return strlen(str_buf);
168: }
```

File: sdm/common/message/SDMReady.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMReady.h"
4: #include "../UDPcom.h"
5:
6: SDMReady::SDMReady():destination(), source()
7: {
8:     MsgName = SDM_Ready;
9:     total_length = 20;
10: }
11:
12: long SDMReady::Send()
13: {
14:     if (destination.getPort() == PORT_PM)
15:         return SendTM();
16:     return SendDM();
17: }
18:
19: long SDMReady::Send(const SDMComponent_ID& destination)
20: {
21:     return SendTo(destination);
22: }
23:
24: long SDMReady::Sendtcp(const SDMComponent_ID& destination)
25: {
26:     return SendTCP(destination.getAddress(),destination.getPort());
27: }
28:
29: //addr must be in network byte order
30: long SDMReady::SendBCast(long addr, unsigned short dest_port)
31: {
32:     return SendBroadcast(addr,dest_port);
33: }
34:
35: long SDMReady::Marshal(char* buf)
36: {
37:     int cur = HEADER_SIZE;
38:     cur += source.Marshal(buf, cur);
39:     cur += destination.Marshal(buf,cur);
```

```

40: msg_length = cur - HEADER_SIZE;
41: MarshalHeader(buf);
42: return cur;
43: }
44:
45: long SDMReady::Unmarshal(const char* buf)
46: {
47: int cur;
48: cur = UnmarshalHeader(buf);
49: if(cur == SDM_INVALID_MESSAGE)
50: {
51:     return SDM_INVALID_MESSAGE;
52: }
53: if(total_length!=msg_length)
54: {
55:     return SDM_INVALID_MESSAGE;
56: }
57: cur += source.Unmarshal(buf, cur);
58: cur += destination.Unmarshal(buf,cur);
59: #ifdef BUILD_WITH_MESSAGE_LOGGING
60: Logger.MessageReceived(*this);
61: #endif
62: return HEADER_SIZE+msg_length;
63: }
64: int SDMReady::MsgToString(char *str_buf, int length) const
65: {
66: char dest_id[40];
67: char source_id[40];
68:
69: if (length < 8096)
70:     return 0;
71: destination.IDToString(dest_id, 40);
72: source.IDToString(source_id, 40);
73: #ifdef WIN32
74: sprintf(str_buf,
75: #else
76: snprintf(str_buf,length,
77: #endif
78:     "SDMReady %ld:%ld %ld bytes, Source: %s Dest: %s", sec, subsec, msg_length, source_id,
    dest_id);
79: return (int)strlen(str_buf);

```


80: }

File: sdm/common/message/SDMCode.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMCode.h"
4: #include "../checksum/checksum.h"
5: #include "../UDPcom.h"
6:
7:
8: SDMCode::SDMCode() : seq_num(0), num_segments(0), code_length(0), c_sum(0)
9: {
10:  MsgName = SDM_Code;
11:  filename[0] = '\0';
12:  total_length = 9;
13: }
14:
15: long SDMCode::Marshal(char* buf)
16: {
17:  int cur;
18:  cur = HEADER_SIZE;
19:  PUT_USHORT(&buf[cur], seq_num);
20:  cur += sizeof(seq_num);
21:  PUT_USHORT(&buf[cur], num_segments);
22:  cur += sizeof(num_segments);
23:  //
24:  // Copy the filename
25:  size_t uiCurLength = strlen(filename);
26:  strcpy(buf + cur, filename);
27:  cur += (int)uiCurLength + 1;
28:  //
29:  // Copy the code section
30:  memcpy (&buf[cur], &code, code_length);
31:  cur += code_length;
32:
33:  c_sum = checksum(code, code_length);
34:  PUT_INT (&buf[cur], c_sum);
35:  cur += sizeof (c_sum);
36:  msg_length = cur - HEADER_SIZE;
37:  MarshalHeader(buf);
38:  return cur;
39: }
```

```

40:
41: long SDMCode::Unmarshal(const char* buf)
42: {
43: int cur;
44: cur = UnmarshalHeader(buf);
45: if (cur == SDM_INVALID_MESSAGE)
46: {
47:     return SDM_INVALID_MESSAGE;
48: }
49: if(total_length>msg_length)
50: {
51:     return SDM_INVALID_MESSAGE;
52: }
53: seq_num = GET_USHORT(&buf[cur]);
54: cur += sizeof(seq_num);
55: num_segments = GET_USHORT(&buf[cur]);
56: cur += sizeof(num_segments);
57: //
58: // Get the filename
59: size_t uiCurLength = strlen(buf + cur);
60: strncpy(filename, buf + cur, sizeof(filename));
61: if (uiCurLength > sizeof(filename) - 1)
62:     filename[sizeof(filename) - 1] = '\0';
63: cur += (int)uiCurLength + 1;
64:
65: memcpy(&code, &buf[cur], msg_length - (strlen(filename)+1) - 8);
66: cur += msg_length - (unsigned int)(strlen(filename)+1) - 8;
67: code_length = msg_length - (unsigned int)(strlen(filename)+1) - 8;
68: c_sum = GET_INT(&buf[cur]);
69: cur += sizeof(c_sum);
70: #ifdef BUILD_WITH_MESSAGE_LOGGING
71: Logger.MessageReceived(*this);
72: #endif
73: return cur;
74: }
75:
76: bool SDMCode::isCorrupt()
77: {
78: long curr_csum = c_sum;
79: long msg_csum = checksum(code, code_length);
80: return curr_csum == msg_csum ? false : true;

```

```
81: }
82: int SDMCode::MsgToString(char *str_buf, int length) const
83: {
84: if (length < 8096)
85:     return 0;
86: #ifdef WIN32
87: sprintf(str_buf,
88: #else
89: snprintf(str_buf,length,
90: #endif
91:     "SDMCode %ld:%ld %ld bytes, SeqNum: %d NumSegments: %d Filename: %s [CODE
SECTION] Checksum: %d", sec, subsec, msg_length, seq_num, num_segments, filename,c_sum);
92: return (int)strlen(str_buf);
93: }
```

File: sdm/common/message/SDMReqReg.h

```
1: #ifndef __SDM_REQREG_H_
2: #define __SDM_REQREG_H_
3:
4: #include "SDMmessage.h"
5:
6: #define SDM_REQREG_CURRENT          0
7: #define SDM_REQREG_CURRENT_AND_FUTURE      5
8: #define SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS  7
9: #define SDM_REQREG_CANCEL          11
10:
11: #define QUALLIST_MAX_SIZE          1024
12:
13: class SDMLIB_API SDMReqReg: public SDMmessage
14: {
15: public :
16:     SDMComponent_ID source;          //optional componentID of the xTEDS provider to search
17:     SDMComponent_ID destination;     //the subscriber
18:     unsigned char reply;             //value indicating the style of reply desired see #defines above
19:     unsigned short id;               //id of this request
20:     char device[XTEDS_MAX_ITEM_NAME_SIZE]; //optional name of the device or application
        to search
21:     char interface[XTEDS_MAX_ITEM_NAME_SIZE]; //optional name of the interface to search
22:     char item_name[XTEDS_MAX_ITEM_NAME_SIZE]; //optional buffer containing item name to
        look for
23:     char quallist[QUALLIST_MAX_SIZE]; //optional buffer containing qualifications for item
24:     SDMReqReg();
25:     long Send();
26:     long Recv(long port); SDM_DEPRECATED
27:     long Marshal(char* buf);
28:     long Unmarshal(const char* buf);
29:     int MsgToString(char *buf, int length) const;
30: };
31:
32: #endif
```

File: sdm/common/message/SDMHeartbeat.cpp

```
1: #include "SDMHeartbeat.h"
2: #include "../Time/SDMTime.h"
3: #include <string.h>
4: #include <stdio.h>
5:
6:
7: /*
8:  * Send a heartbeat message to the local Process Manager. The PM knows who the application
9:  * is by way of application PID in the sensor ID field of the message. Therefore, this
10:  * can't be called until SDMInit() has been.
11:  */
12: SDMLIB_API void SendHeartbeat()
13: {
14:     const unsigned int MIN_SEND_INTERVAL = 20; // Minimum time to wait before re-sending, in
seconds
15:     unsigned int uiCurSeconds = 0, uiTemp = 0;
16:     static unsigned int uiLastSendTime = 0;
17:     SDM_GetTime(&uiCurSeconds, &uiTemp);
18:
19:     // If the previous heartbeat was sent within the minimum interval, don't send
20:     if (uiCurSeconds - uiLastSendTime < MIN_SEND_INTERVAL && uiLastSendTime != 0)
21:         return;
22:
23:     SDMHeartbeat msgHeartbeat;
24:     SDMComponent_ID PmId;
25:
26:     msgHeartbeat.source.setSensorID(PID);
27:     PmId.setAddress(ADDR_LOCAL_HOST);
28:     PmId.setPort(PORT_PM);
29:     msgHeartbeat.SendTo(PmId);
30:
31:     uiLastSendTime = uiCurSeconds;
32: }
33:
34: SDMHeartbeat::SDMHeartbeat():source()
35: {
36:     MsgName = SDM_Heartbeat;
37:     total_length = 10;
38: }
```

```

39:
40: long SDMHeartbeat::Marshal(char *buf)
41: {
42:     int cur = HEADER_SIZE;
43:     cur += source.Marshal(buf, cur);
44:     msg_length = cur-HEADER_SIZE;
45:     MarshalHeader(buf);
46:     return cur;
47: }
48:
49: long SDMHeartbeat::Unmarshal(const char * buf)
50: {
51:     int cur = UnmarshalHeader(buf);
52:     if (cur==SDM_INVALID_MESSAGE)
53:     {
54:         return SDM_INVALID_MESSAGE;
55:     }
56:     if (total_length != msg_length)
57:     {
58:         return SDM_INVALID_MESSAGE;
59:     }
60:     cur += source.Unmarshal(buf, cur);
61: #ifdef BUILD_WITH_MESSAGE_LOGGING
62:     Logger.MessageReceived(*this);
63: #endif
64:     return cur;
65: }
66:
67: int SDMHeartbeat::MsgToString(char * str_buf, int length) const
68: {
69:     char source_id[40];
70:
71:     if (length < 8096)
72:         return 0;
73:     source.IDToString(source_id, 40);
74: #ifdef WIN32
75:     sprintf(str_buf,
76: #else
77:     snprintf(str_buf,length,
78: #endif
79:         "SDMHeartbeat %ld:%ld %ld bytes, Source: %s", sec, subsec, msg_length, source_id);

```

```
80: return (int)strlen(str_buf);  
81: }
```


File: sdm/common/message/SDMError.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "SDMError.h"
4: #include "../UDPcom.h"
5: #include "SDMmessage.h"
6:
7: SDMError::SDMError():source(),error(0),msg_id(0,0),error_msg()
8: {
9:     MsgName = SDM_Error;
10:    total_length = 13;
11:    error_msg[0] = '\0';
12: }
13:
14: long SDMError::Send()
15: {
16:    return SendDM();
17: }
18:
19: long SDMError::Send(const SDMComponent_ID& destination)
20: {
21:    return SendTo(destination);
22: }
23:
24: long SDMError::Recv(long port)
25: {
26:    int result;
27:    result = RecvFrom(port);
28:    if (result <= 0)
29:        return SDM_MESSAGE_RECV_ERROR;
30:    return result;
31: }
32:
33: long SDMError::Marshal(char* buf)
34: {
35:    int cur = HEADER_SIZE;
36:    cur += source.Marshal(buf,cur);
37:    cur += msg_id.Marshal(buf, cur);
38:    PUT_UCHAR (&buf[cur], error);
39:    cur += sizeof(error);
```

```

40:
41: unsigned int uiCurLength = (unsigned int)strlen(error_msg);
42: strncpy(buf + cur, error_msg, uiCurLength + 1);
43: cur += uiCurLength + 1;
44:
45: msg_length = cur - HEADER_SIZE;
46: MarshalHeader(buf);
47: return HEADER_SIZE+msg_length;
48: }
49:
50: long SDLError::Unmarshal(const char* buf)
51: {
52: int cur;
53: cur = UnmarshalHeader(buf);
54: if(cur==SDM_INVALID_MESSAGE)
55: {
56:     return SDM_INVALID_MESSAGE;
57: }
58: if(total_length>msg_length)
59: {
60:     return SDM_INVALID_MESSAGE;
61: }
62: cur += source.Unmarshal(buf,cur);
63: cur += msg_id.Unmarshal(buf, cur);
64: error = GET_UCHAR (&buf[cur]);
65: cur += sizeof(error);
66:
67: unsigned int uiMsgLength = (unsigned int)strlen(buf + cur);
68: strncpy(error_msg, buf + cur, sizeof(error_msg));
69: if (uiMsgLength > sizeof(error_msg) - 1)
70:     error_msg[sizeof(error_msg) - 1] = '\0';
71:
72: #ifdef BUILD_WITH_MESSAGE_LOGGING
73: Logger.MessageReceived(*this);
74: #endif
75: return HEADER_SIZE+msg_length;
76: }
77: int SDLError::MsgToString(char *str_buf, int length) const
78: {
79: char source_id[40];
80: char message_id[30];

```

```
81:
82: if (length < 8096)
83:     return 0;
84: source.IDToString(source_id, 40);
85: msg_id.IDToString(message_id, 30);
86: #ifdef WIN32
87: sprintf(str_buf,
88: #else
89: snprintf(str_buf,length,
90: #endif
91:     "SDMError %ld:%ld %ld bytes, Source: %s %s Error: %d", sec, subsec, msg_length, source_id,
message_id, error);
92: return (int)strlen(str_buf);
93: }
```

File: sdm/common/message/SDMCommand.h

```
1: #ifndef __SDM_COMMAND_H
2: #define __SDM_COMMAND_H
3:
4: #include "SDMService.h"
5:
6: class SDMLIB_API SDMCommand:public SDMService
7: {
8: public:
9:     SDMMMessage_ID fault_id;    //unique id of possible fault message, may be 0 for no fault
10:     SDMCommand();
11:     long Marshal(char* buf);
12:     long Unmarshal(const char* buf);
13:     long Send();
14:     long Send(const SDMComponent_ID& destination);
15:     long Recv(long port); SDM_DEPRECATED
16:     int MsgToString(char *str_buf, int length) const;
17: };
18:
19: #endif
```

File: sdm/common/message/SDMDMLLeader.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "SDMDMLLeader.h"
4:
5: SDMDMLLeader::SDMDMLLeader(): source(),running_flag(0)
6: {
7:     MsgName = SDM_DMLeader;
8:     total_length = 11;
9: }
10: SDMDMLLeader::~SDMDMLLeader()
11: {
12: }
13:
14: long SDMDMLLeader::Marshal(char *buf)
15: {
16:     int cur = HEADER_SIZE;
17:     cur += source.Marshal(buf, cur);
18:     PUT_CHAR(&buf[cur], running_flag);
19:     cur++;
20:     msg_length = cur - HEADER_SIZE;
21:     MarshalHeader(buf);
22:     return cur;
23: }
24:
25: long SDMDMLLeader::Unmarshal(const char *buf)
26: {
27:     int cur = 0;
28:     cur = UnmarshalHeader(buf);
29:     if (cur == SDM_INVALID_MESSAGE)
30:         return SDM_INVALID_MESSAGE;
31:     if (total_length != msg_length)
32:         return SDM_INVALID_MESSAGE;
33:     cur += source.Unmarshal(buf,cur);
34:     running_flag = GET_CHAR(&buf[cur]);
35:     cur++;
36: #ifdef BUILD_WITH_MESSAGE_LOGGING
37:     Logger.MessageReceived(*this);
38: #endif
39:     return cur;
```

```

40: }
41:
42: //SDMDMLLeader messages are sent via Broadcast
43: long SDMDMLLeader::Send()
44: {
45:     return SendBroadcast();
46: }
47: int SDMDMLLeader::MsgToString(char *str_buf, int length) const
48: {
49:     char source_id[40];
50:
51:     if (length < 8096)
52:         return 0;
53:     source.IDToString(source_id, 40);
54: #ifdef WIN32
55:     sprintf(str_buf,
56: #else
57:     snprintf(str_buf,length,
58: #endif
59:         "SDMDMLLeader %ld:%ld %ld bytes, Source: %s running_flag: %c", sec, subsec, msg_length,
        source_id, running_flag);
60:     return (int)strlen(str_buf);
61: }

```

File: sdm/common/message/SDMTaskFinished.h

```
1: #ifndef __SDM_TASK_FINISHED_H_
2: #define __SDM_TASK_FINISHED_H_
3:
4: #include "SDMmessage.h"
5:
6: //This class is intended only for use by SDM core components
7:
8: class SDMLIB_API SDMTaskFinished : public SDMmessage
9: {
10: public:
11:     SDMComponent_ID source;           //The PM on which the finished task resides
12:     int result;                       //The result value of the task running
13:     char filename[MAX_FILENAME_SIZE]; //The name of the task finished
14:     int pid;
15:     SDMTaskFinished();
16:     long Send();
17:     long Marshal (char* buf);
18:     long Unmarshal (const char* buf);
19:     int MsgToString(char *str_buf, int length) const;
20: };
21:
22:
23: #endif
```

File: sdm/common/message/SDMVarReq.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMVarReq.h"
4:
5: SDMVarReq::SDMVarReq():source(), destination(), reply(SDM_VARREQ_CURRENT), msg_id(),
id(0)
6: {
7:   MsgName = SDM_VarReq;
8:   total_length = 25;
9:   variable[0] = '\0';
10: }
11:
12: long SDMVarReq::Send()
13: {
14:   return SendDM();
15: }
16:
17: long SDMVarReq::Marshal(char *buf)
18: {
19:   int cur = HEADER_SIZE;
20:   cur += source.Marshal(buf, cur);
21:   cur += destination.Marshal(buf, cur);
22:   PUT_UCHAR(&buf[cur], reply);
23:   cur += sizeof(reply);
24:   cur += msg_id.Marshal(buf, cur);
25:   PUT_USHORT(buf+cur, id);
26:   cur += sizeof(id);
27:   strcpy(buf+cur, variable);      //Copy the variable string field
28:   cur += strlen(variable) + 1;    //Cur is now the strlen plus null terminator
29:   msg_length = cur - HEADER_SIZE;
30:   MarshalHeader(buf);
31:   return cur;
32: }
33:
34: long SDMVarReq::Unmarshal(const char *buf)
35: {
36:   int cur = UnmarshalHeader(buf);
37:   if (cur == SDM_INVALID_MESSAGE || total_length > msg_length)
38:     return SDM_INVALID_MESSAGE;
```



```

39: cur += source.Unmarshal(buf, cur);
40: cur += destination.Unmarshal(buf, cur);
41: reply = GET_UCHAR (&buf[cur]);
42: cur += sizeof(reply);
43: cur += msg_id.Unmarshal(buf, cur);
44: id = GET_USHORT(buf+cur);
45: cur += sizeof(id);
46: strcpy(variable, buf+cur);
47: cur += strlen(variable) + 1;
48: #ifdef BUILD_WITH_MESSAGE_LOGGING
49: Logger.MessageReceived(*this);
50: #endif
51: return cur;
52: }
53:
54: int SDMVarReq::MsgToString(char *str_buf, int length) const
55: {
56: char source_id[40];
57: char dest_id[40];
58: char message_id[40];
59:
60: if (length < 8096)
61:     return 0;
62: source.IDToString(source_id, 40);
63: destination.IDToString(dest_id, 40);
64: msg_id.IDToString(message_id, 30);
65: #ifdef WIN32
66: sprintf(str_buf,
67: #else
68: snprintf(str_buf,length,
69: #endif
70:     "SDMVarReq %ld:%ld %ld bytes, Source: %s Dest: %s MsgID: %s id: %hu variable: %s", sec,
subsec, msg_length, source_id, dest_id, message_id, id, variable);
71: return strlen(str_buf);
72: }

```

File: sdm/common/message/SDMCancel.cpp

```
1: #include "SDMCancel.h"
2: #include <string.h>
3: #include <stdio.h>
4:
5: SDMCancel::SDMCancel():source(),destination(),msg_id(0,0)
6: {
7:     MsgName = SDM_Cancel;
8:     total_length = 22;
9: }
10:
11: long SDMCancel::Send()
12: {
13:     return SendDM();
14: }
15:
16: long SDMCancel::Marshal(char* buf)
17: {
18:     int cur;
19:     cur = HEADER_SIZE;
20:     cur += source.Marshal(buf,cur);
21:     cur += destination.Marshal(buf,cur);
22:     cur += msg_id.Marshal(buf, cur);
23:     msg_length = cur-HEADER_SIZE;
24:     MarshalHeader(buf);
25:     return cur;
26: }
27:
28: long SDMCancel::Unmarshal(const char* buf)
29: {
30:     int cur;
31:     cur = UnmarshalHeader(buf);
32:     if(cur==SDM_INVALID_MESSAGE)
33:     {
34:         return SDM_INVALID_MESSAGE;
35:     }
36:     if(total_length!=msg_length)
37:     {
38:         return SDM_INVALID_MESSAGE;
39:     }
```

```

40: cur += source.Unmarshal(buf,cur);
41: cur += destination.Unmarshal(buf,cur);
42: cur += msg_id.Unmarshal(buf, cur);
43: #ifdef BUILD_WITH_MESSAGE_LOGGING
44: Logger.MessageReceived(*this);
45: #endif
46: return cur;
47: }
48: /*
49:  * MsgToString() puts the message into str_buf into human readable form, returning the number of
  characters copied.
50: */
51: int SDMCancel::MsgToString(char *str_buf, int length) const
52: {
53: char source_id[40];
54: char dest_id[40];
55: char message_id[30];
56:
57: if (length < 8096)
58:     return 0;
59: source.IDToString(source_id, 40);
60: destination.IDToString(dest_id, 40);
61: msg_id.IDToString(message_id, 30);
62: #ifdef WIN32 // cleanup - remove duplicate code for maintainability
63: sprintf(str_buf,
64: #else
65: snprintf(str_buf,length,
66: #endif
67:     "SDMCancel %ld:%ld %ld bytes, Source: %s Dest: %s %s", sec, subsec, msg_length, source_id,
  dest_id, message_id);
68: return (int)strlen(str_buf);
69: }

```

File: sdm/common/message/SDMRegInfo.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMRegInfo.h"
4: #include "../UDPcom.h"
5:
6: SDMRegInfo::SDMRegInfo():id(0),type(0),source(),msg_id(0,0),emptyflag(false)
7: {
8:     MsgName = SDM_RegInfo;
9:     msg_def[0] = '\0';
10:    xTEDS_section[0] = '\0';
11:    total_length = 16;
12: }
13:
14: long SDMRegInfo::Send(const SDMComponent_ID& destination)
15: {
16:     int result;
17:     emptyflag = false;
18:     result = SendTo(destination);
19:     if (result <= 0)
20:         return SDM_MESSAGE_SEND_ERROR;
21:     return result;
22: }
23:
24: long SDMRegInfo::Recv(long port)
25: {
26:     int result;
27:     result = RecvFrom(port);
28:     if (result <= 0)
29:         return SDM_MESSAGE_RECV_ERROR;
30:     if (result == 1)
31:         return SDM_NO_FURTHER_DATA_PROVIDER;
32:     return result;
33: }
34:
35: long SDMRegInfo::Marshal(char* buf)
36: {
37:     int cur;
38:     if(emptyflag == true)
39: {
```

```

40:     cur = MarshalEmpty(buf);
41:     return cur;
42: }
43: cur = HEADER_SIZE;
44: cur += source.Marshal(buf,cur);
45: cur += msg_id.Marshal(buf, cur);
46: PUT_USHORT (&buf[cur], id);
47: cur += sizeof(id);
48: PUT_UCHAR (&buf[cur], type);
49: cur += sizeof(type);
50: strcpy(buf+cur, msg_def);
51: cur += (int)strlen(msg_def) + 1;    //Add the string length plus the null terminator
52: strcpy(buf+cur, xTEDS_section);
53: cur += (int)strlen(xTEDS_section) + 1; //Add the string length plus the null terminator
54: msg_length = cur - HEADER_SIZE;
55: MarshalHeader(buf);
56: return cur;
57: }
58:
59: long SDMRegInfo::Unmarshal(const char* buf)
60: {
61: int cur;
62: cur = UnmarshalHeader(buf);
63: if(cur==SDM_INVALID_MESSAGE)
64: {
65:     return SDM_INVALID_MESSAGE;
66: }
67: if(msg_length == 0)
68:     return SDM_NO_FURTHER_DATA_PROVIDER;
69: if(total_length>msg_length)
70: {
71:     return SDM_INVALID_MESSAGE;
72: }
73: cur += source.Unmarshal(buf,cur);
74: cur += msg_id.Unmarshal(buf, cur);
75: id = GET_USHORT (&buf[cur]);
76: cur += sizeof(id);
77: type = GET_UCHAR (&buf[cur]);
78: cur += sizeof(type);
79: strcpy(msg_def, buf+cur);
80: cur += (int)strlen(msg_def) + 1;    //Add strlen plus null terminator

```

```

81: strcpy(xTEDS_section, buf+cur);
82: cur += (int)strlen(xTEDS_section) + 1; //Add strlen plus null terminator
83: #ifdef BUILD_WITH_MESSAGE_LOGGING
84: Logger.MessageReceived(*this);
85: #endif
86: return cur;
87: }
88:
89: long SDMRegInfo::MarshalEmpty(char* buf)
90: {
91: int cur;
92: msg_length = 0;
93: cur = MarshalHeader(buf);
94: return cur;
95: }
96:
97: long SDMRegInfo::SendEmpty(const SDMComponent_ID& destination)
98: {
99: int result;
100:     emptyflag = true;
101:     result = SendTo(destination);
102:     if (result <= 0)
103:         return SDM_MESSAGE_SEND_ERROR;
104:     return result;
105: }
106:
107: int SDMRegInfo::MsgToString(char *str_buf, int length) const
108: {
109:     char source_id[40];
110:     char message_id[30];
111:
112:     if (length < 8096)
113:         return 0;
114:     source.IDToString(source_id, 40);
115:     msg_id.IDToString(message_id, 30);
116: #ifdef WIN32
117:     sprintf(str_buf,
118: #else
119:     snprintf(str_buf, length,
120: #endif

```

```
121:         "SDMRegInfo %ld:%ld %ld bytes, Source: %s %s ID: %d Type: %d MsgDef: %s", sec,  
subsec, msg_length, source_id, message_id, id, type, msg_def);  
122:     return (int)strlen(str_buf);  
123: }
```

File: sdm/common/message/SDMReqReg.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMReqReg.h"
4:
5: SDMReqReg::SDMReqReg():source(),destination(),reply(SDM_REQREG_CURRENT),id(0)
6: {
7:     MsgName = SDM_ReqReg;
8:     device[0] = interface[0] = item_name[0] = quallist[0] = '\0';
9:     total_length = 27;
10: }
11:
12: long SDMReqReg::Send()
13: {
14:     return SendDM();
15: }
16:
17: long SDMReqReg::Recv(long port)
18: {
19:     int result;
20:     result = RecvFrom(port);
21:     if (result <= 0)
22:         return SDM_MESSAGE_RECV_ERROR;
23:     return result;
24: }
25:
26: long SDMReqReg::Marshal(char* buf)
27: {
28:     int cur;
29:     long device_length;
30:     long interface_length;
31:     long item_name_msg_length;
32:     long quallist_msg_length;
33:
34:     device_length = (long)strlen(device);
35:     interface_length = (long)strlen(interface);
36:     item_name_msg_length = (long)strlen(item_name);
37:     quallist_msg_length = (long)strlen(quallist);
38:     cur = HEADER_SIZE;
39:     cur += source.Marshal(buf,cur);
```



```

40: cur += destination.Marshal(buf,cur);
41: PUT_UCHAR (&buf[cur], reply);
42: cur += sizeof(reply);
43: PUT_USHORT (&buf[cur], id);
44: cur += sizeof(id);
45: memcpy(buf+cur,device,device_length);
46: cur += device_length;
47: buf[cur] = '\0';
48: cur++;
49: memcpy(buf+cur,interface,interface_length);
50: cur += interface_length;
51: buf[cur] = '\0';
52: cur++;
53: memcpy(buf+cur,item_name,item_name_msg_length);
54: cur += item_name_msg_length;
55: buf[cur] = '\0';
56: cur++;
57: memcpy(buf+cur,quallist,quallist_msg_length);
58: cur += quallist_msg_length;
59: buf[cur] = '\0';
60: cur++;
61: msg_length = cur - HEADER_SIZE;
62: MarshalHeader(buf);
63: return cur;
64: }
65:
66: long SDMReqReg::Unmarshal(const char* buf)
67: {
68: int cur;
69: unsigned int uiCurLength;
70:
71: //unmarshal header
72: cur = UnmarshalHeader(buf);
73: if(cur == SDM_INVALID_MESSAGE)
74: {
75:     return SDM_INVALID_MESSAGE;
76: }
77: if(total_length>msg_length)
78: {
79:     return SDM_INVALID_MESSAGE;
80: }

```

```

81: cur += source.Unmarshal(buf,cur);
82: cur += destination.Unmarshal(buf,cur);
83: reply = GET_UCHAR (&buf[cur]);
84: cur += sizeof(reply);
85: id = GET_USHORT (&buf[cur]);
86: cur += sizeof(id);
87: //
88: // Get the device name
89: uiCurLength = (unsigned int)strlen(buf+cur);
90: strncpy(device, buf+cur, sizeof(device));
91: if (uiCurLength > sizeof(device) - 1)
92:     device[sizeof(device) - 1] = '\0';
93: cur += uiCurLength + 1;
94: //
95: // Get the interface name
96: uiCurLength = (unsigned int)strlen(buf+cur);
97: strncpy(interface, buf+cur, sizeof(interface));
98: if (uiCurLength > sizeof(interface) - 1)
99:     interface[sizeof(interface) - 1] = '\0';
100: cur += uiCurLength + 1;
101: //
102: // Get the item name
103: uiCurLength = (unsigned int)strlen(buf+cur);
104: strncpy(item_name, buf+cur, sizeof(item_name));
105: if (uiCurLength > sizeof(item_name) - 1)
106:     item_name[sizeof(item_name) - 1] = '\0';
107: cur += uiCurLength + 1;
108: //
109: // Get the qual list
110: uiCurLength = (unsigned int)strlen(buf+cur);
111: strncpy(quallist, buf+cur, sizeof(quallist));
112: if (uiCurLength > sizeof(quallist) - 1)
113:     quallist[sizeof(quallist) - 1] = '\0';
114: cur += uiCurLength + 1;
115:
116: #ifdef BUILD_WITH_MESSAGE_LOGGING
117:     Logger.MessageReceived(*this);
118: #endif
119:     return cur;
120: }
121: int SDMReqReg::MsgToString(char *str_buf, int length) const

```

```

122: {
123:   char dest_id[40];
124:   char provider_id[40];
125:
126:   if (length < 8096)
127:       return 0;
128:   destination.IDToString(dest_id, 40);
129:   source.IDToString(provider_id, 40);
130: #ifdef WIN32
131:   sprintf(str_buf,
132: #else
133:   snprintf(str_buf, length,
134: #endif
135:           "SDMReqReg %ld:%ld %ld bytes, Dest: %s Reply: %d ID: %d Provider: %s Device: %s
Interface: %s ItemName: %s QualList: %s", sec, subsec, msg_length, dest_id, reply, id, provider_id,
device, interface, item_name, quallist);
136:   return (int)strlen(str_buf);
137: }

```

File: sdm/common/message/SDMReqCode.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMReqCode.h"
4:
5: SDMReqCode::SDMReqCode():source(), seq_num(0), version(0)
6: {
7:     MsgName = SDM_ReqCode;
8:     filename[0] = '\0';
9:     total_length = 17;
10: }
11:
12: long SDMReqCode::Send()
13: {
14:     return SendTM();
15: }
16:
17: long SDMReqCode::Recv(long port)
18: {
19:     int result;
20:     result = RecvFrom(port);
21:     if (result <= 0)
22:         return SDM_MESSAGE_RECV_ERROR;
23:     return result;
24: }
25:
26: long SDMReqCode::Marshal(char* buf)
27: {
28:     int cur;
29:     int filename_length;
30:     cur = HEADER_SIZE;
31:     filename_length = (int)strlen(filename) + 1;
32:     cur += source.Marshal(buf,cur);
33:     PUT_USHORT(&buf[cur], seq_num);
34:     cur += sizeof(seq_num);
35:     PUT_INT(&buf[cur], version);
36:     cur += sizeof(version);
37:     memcpy (&buf[cur], &filename, filename_length);
38:     cur += filename_length;
39:     msg_length = cur - HEADER_SIZE;
```

```

40: MarshalHeader(buf);
41: return cur;
42: }
43:
44: long SDMReqCode::Unmarshal(const char* buf)
45: {
46: int cur;
47: unsigned int filename_length;
48: int source_size;
49: cur = UnmarshalHeader(buf);
50: if(cur == SDM_INVALID_MESSAGE)
51: {
52:     return SDM_INVALID_MESSAGE;
53: }
54: if(total_length>msg_length)
55: {
56:     return SDM_INVALID_MESSAGE;
57: }
58: source_size = source.Unmarshal(buf, cur);
59: filename_length = msg_length - source_size - sizeof(seq_num);
60: cur += source_size;
61: seq_num = GET_USHORT(&buf[cur]);
62: cur += sizeof(seq_num);
63: version = GET_INT(&buf[cur]);
64: cur += sizeof(version);
65: //
66: // Get the filename
67: filename_length = (unsigned int)strlen(buf + cur);
68: strncpy(filename, buf + cur, sizeof(filename));
69: if (filename_length > sizeof(filename) - 1)
70:     filename[sizeof(filename) - 1] = '\0';
71: cur += filename_length + 1;
72:
73: #ifdef BUILD_WITH_MESSAGE_LOGGING
74: Logger.MessageReceived(*this);
75: #endif
76: return cur;
77: }
78: int SDMReqCode::MsgToString(char *str_buf, int length) const
79: {
80: char source_id[40];

```

```
81:
82: if (length < 8096)
83:     return 0;
84: source.IDToString(source_id, 40);
85: #ifdef WIN32
86: sprintf(str_buf,
87: #else
88: snprintf(str_buf,length,
89: #endif
90:     "SDMReqCode %ld:%ld %ld bytes, Source: %s Filename: %s seq_num: %hd version %d", sec,
subsec, msg_length, source_id, filename, seq_num, version);
91: return (int)strlen(str_buf);
92: }
```

File: sdm/common/message/Makefile

```
1: #SDM message classes
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: SDMClasses = SDMAck SDMCancel SDMCancelxTEDS SDMCode SDMCommand SDMConsume
SDMComponent_ID SDMData SDMDeletesub SDMDMLLeader SDMElection SDMError SDMHeartbeat
SDMKill SDMmessage SDMMessage_ID SDMPostTask SDMReady SDMRegInfo SDMRegPM
SDMReqCode SDMReqReg SDMReqxTEDS SDMSearch SDMSearchReply SDMSerreqst SDMService
SDMSubreqst SDMTask SDMTaskError SDMTaskFinished SDMTat SDMVarInfo SDMVarReq
SDMxTEDS SDMxTEDSInfo SDMHello SDMRegister SDMID
7:
8: .PHONY: all clean distclean
9:
10: all: $(addsuffix .o,$(SDMClasses))
11:
12: %.o:    %.cpp %.h
13: $(CXX) $(CXXFLAGS) $(MESSAGELOGGINGFLAGS) -fPIC -c $<
14:
15: clean:
16: rm -f *.o *~
17:
18: distclean:    clean
```

File: sdm/common/message/SDMSearch.h

```
1: #ifndef __SDM_SEARCH_H_
2: #define __SDM_SEARCH_H_
3:
4: #include "SDMmessage.h"
5:
6: #define SDM_SEARCH_CURRENT          0
7: #define SDM_SEARCH_CURRENT_AND_FUTURE 5
8: #define SDM_SEARCH_CANCEL          11
9:
10: class SDMLIB_API SDMSearch: public SDMmessage
11: {
12: public :
13:     SDMComponent_ID source;          //optional component_id of the xTEDS provider
14:     SDMComponent_ID destination;     //the id of the message issuer
15:     unsigned char reply;             //style of reply desired (see above)
16:     unsigned short id;               //id of this request
17:     char reg_expr[BUFSIZE-33];       //buffer containing regular expression
18:     SDMSearch();
19:     long Send();
20:     long Recv(long port); SDM_DEPRECATED
21:     long Marshal(char* buf);
22:     long Unmarshal(const char* buf);
23:     int MsgToString(char *buf, int length) const;
24: };
25:
26: #endif
```


File: sdm/common/message/SDMRegPM.h

```
1: #ifndef __SDM_REG_PM_H_
2: #define __SDM_REG_PM_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMComponent_ID.h"
6:
7: //This is intended only for use by SDM core components
8:
9: class SDMLIB_API SDMRegPM: public SDMmessage
10: {
11: public :
12:     SDMComponent_ID source;          //The PM registering with the TM
13:     unsigned short resources;        //an enumeration of the resources of a processor
14:     SDMRegPM();
15:     long Send();
16:     long Send(const SDMComponent_ID& destination);
17:     long Marshal(char* buf);
18:     long Unmarshal(const char* buf);
19:     int MsgToString(char *buf, int length) const;
20: };
21:
22: #endif
```

File: sdm/common/message/SDMTask.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMTask.h"
4: #include "../UDPcom.h"
5:
6: SDMTask::SDMTask():source(),priority(0),pid(0),version(0)
7: {
8:     MsgName = SDM_Task;
9:     filename[0] = '\0';
10:    total_length = 10;
11: }
12:
13: long SDMTask::Send(const SDMComponent_ID& destination)
14: {
15:    return SendTo(destination);
16: }
17:
18: long SDMTask::Marshal(char* buf)
19: {
20:    int cur;
21:    long filename_msg_length;
22:
23:    filename_msg_length = (long)strlen(filename);
24:    cur = HEADER_SIZE;
25:    cur += source.Marshal(buf,cur);
26:    PUT_CHAR(&buf[cur],priority);
27:    cur += sizeof(priority);
28:    PUT_LONG(&buf[cur],pid);
29:    cur += sizeof(pid);
30:    PUT_INT(&buf[cur], version);
31:    cur += sizeof(version);
32:    memcpy(buf+cur,&filename,filename_msg_length);
33:    cur += filename_msg_length;
34:    buf[cur]=0;
35:    cur++;
36:    msg_length = cur - HEADER_SIZE;
37:    MarshalHeader(buf);
38:    return cur;
39: }
```

```

40:
41: long SDMTTask::Unmarshal(const char* buf)
42: {
43: int cur;
44: unsigned int uiCurLength;
45: cur = UnmarshalHeader(buf);
46: if(cur == SDM_INVALID_MESSAGE)
47: {
48:     return SDM_INVALID_MESSAGE;
49: }
50: if(total_length>msg_length)
51: {
52:     return SDM_INVALID_MESSAGE;
53: }
54: cur += source.Unmarshal(buf,cur);
55: priority = GET_CHAR (&buf[cur]);
56: cur += sizeof(priority);
57: pid = GET_LONG (&buf[cur]);
58: cur += sizeof(pid);
59: version = GET_INT(&buf[cur]);
60: cur += sizeof(version);
61: //
62: // Get the filename
63: uiCurLength = (unsigned int)strlen(buf+cur);
64: strncpy(filename, buf + cur, sizeof(filename));
65: if (uiCurLength > sizeof(filename) - 1)
66:     filename[sizeof(filename) - 1] = '\0';
67:
68: cur += uiCurLength + 1;
69: #ifdef BUILD_WITH_MESSAGE_LOGGING
70: Logger.MessageReceived(*this);
71: #endif
72: return cur;
73: }
74: int SDMTTask::MsgToString(char *str_buf, int length) const
75: {
76: if (length < 8096)
77:     return 0;
78: #ifdef WIN32
79: sprintf(str_buf,
80: #else

```

```
81: snprintf(str_buf,length,
82: #endif
83:      "SDMTask %ld:%ld %ld bytes, Priority: %hhhd PID: %ld Version: %d Filename: %s", sec,
subsec, msg_length, priority, pid, version, filename);
84: return (int)strlen(str_buf);
85: }
```

File: sdm/common/message/SDMCancelxTEDS.cpp

```
1: #include "SDMCancelxTEDS.h"
2: #include <stdio.h>
3: #include <string.h>
4:
5: SDMCancelxTEDS::SDMCancelxTEDS():source(), fullCancel(0)
6: {
7:     MsgName = SDM_CancelxTEDS;
8:     total_length = 11;
9: }
10:
11: long SDMCancelxTEDS::Send()
12: {
13:     msg_length = 11;
14:     return SendDM();
15: }
16:
17: long SDMCancelxTEDS::Marshal(char* buf)
18: {
19:     int cur;
20:     cur = HEADER_SIZE;
21:     MarshalHeader(buf);
22:     cur += source.Marshal(buf, cur);
23:     memcpy(&buf[cur], &fullCancel, 1);
24:     cur += 1;
25:     msg_length = cur - HEADER_SIZE;
26:
27:     return cur;
28: }
29:
30: long SDMCancelxTEDS::Unmarshal(const char* buf)
31: {
32:     int cur;
33:     cur = UnmarshalHeader(buf);
34:     if(cur == SDM_INVALID_MESSAGE)
35:     {
36:         return SDM_INVALID_MESSAGE;
37:     }
38:     if(msg_length == 10) //If original style SDMCancelxTEDS
39:     {
```

```

40:  cur += source.Unmarshal(buf,cur);
41:  fullCancel = 0;
42: }
43: else
44: {
45:     if(total_length!=msg_length)
46:     {
47:         printf("total length: %i msg legnth: %i \n", total_length, msg_length);
48:         return SDM_INVALID_MESSAGE;
49:     }
50:     cur+= source.Unmarshal(buf,cur);
51:     fullCancel = buf[cur];
52:     cur += 1;
53: }
54: #ifdef BUILD_WITH_MESSAGE_LOGGING
55:  Logger.MessageReceived(*this);
56: #endif
57:
58: return cur;
59: }
60:
61: int SDMCancelxTEDS::MsgToString(char *str_buf, int length) const
62: {
63:  char source_id[40];
64:  if (length < 8096)
65:      return 0;
66:  source.IDToString(source_id, 40);
67:  #ifdef WIN32 // cleanup - remove duplicate code for maintainability
68:  sprintf(str_buf,
69:  #else
70:  snprintf(str_buf, length,
71:  #endif
72:      "SDMCancelxTEDS %ld:%ld %ld bytes, Source: %s", sec, subsec, msg_length, source_id);
73:  return (int)strlen(str_buf);
74: }

```

File: sdm/common/message/SDMRegPM.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "SDMRegPM.h"
4: #include "../UDPcom.h"
5:
6: SDMRegPM::SDMRegPM():source(),resources(0)
7: {
8:     MsgName = SDM_RegPM;
9:     total_length = 12;
10: }
11:
12: long SDMRegPM::Send()
13: {
14:     return SendTM();
15: }
16:
17: long SDMRegPM::Send(const SDMComponent_ID& destination)
18: {
19:     return SendTo(destination);
20: }
21:
22: long SDMRegPM::Marshal(char* buf)
23: {
24:     int cur;
25:     cur = HEADER_SIZE;
26:     cur += source.Marshal(buf,cur);
27:     PUT_USHORT (&buf[cur], resources);
28:     cur += sizeof(resources);
29:     msg_length = cur - HEADER_SIZE;
30:     MarshalHeader(buf);
31:     return cur;
32: }
33:
34: long SDMRegPM::Unmarshal(const char* buf)
35: {
36:     int cur;
37:     cur = UnmarshalHeader(buf);
38:     if(cur == SDM_INVALID_MESSAGE)
39: {
```

```

40:     return SDM_INVALID_MESSAGE;
41: }
42: if(total_length!=msg_length)
43: {
44:     return SDM_INVALID_MESSAGE;
45: }
46: cur += source.Unmarshal(buf,cur);
47: resources = GET_USHORT (&buf[cur]);
48: cur += sizeof(resources);
49: #ifdef BUILD_WITH_MESSAGE_LOGGING
50: Logger.MessageReceived(*this);
51: #endif
52: return cur;
53: }
54: int SDMRegPM::MsgToString(char *str_buf, int length) const
55: {
56: char src_id[40];
57:
58: if (length < 8096)
59:     return 0;
60: source.IDToString(src_id,40);
61: #ifdef WIN32
62: sprintf(str_buf,
63: #else
64: snprintf(str_buf,length,
65: #endif
66:         "SDMRegPM %ld:%ld %ld bytes, Resources: %hu Source: %s", sec, subsec, msg_length,
resources, src_id);
67: return (int)strlen(str_buf);
68: }
69:

```


File: sdm/common/message/SDMTask.h

```
1: #ifndef __SDM_TASK_H_
2: #define __SDM_TASK_H_
3:
4: #include "SDMmessage.h"
5:
6: //This class is intended only for use by SDM core components
7:
8: class SDMLIB_API SDMTask: public SDMmessage
9: {
10: public :
11:     SDMComponent_ID source;           //The source of the SDMTask message
12:     char priority;                    //The priority level of the task (tbd)
13:     char filename[MAX_FILENAME_SIZE]; //The name of the task
14:     long pid;                        //The process id of the task as assigned by the task manager
15:     int version;                     //Some identifying version number, if needed
16:     SDMTask();
17:     long Send(const SDMComponent_ID& destination);
18:     long Marshal(char* buf);
19:     long Unmarshal(const char* buf);
20:     int MsgToString(char *buf, int length) const;
21: };
22:
23: #endif
```

File: sdm/common/message/SDMData.h

```
1: #ifndef __SDM_DATA_H_
2: #define __SDM_DATA_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: typedef struct Listener_Args
8: {
9: int pipe_out;
10: long port;
11: long count;
12: long forward_sensor_id;
13: unsigned char forward_command_id;
14: short (*callback) (char*);
15: } ListenerArgs;
16:
17:
18: class SDMLIB_API SDMData: public SDMmessage
19: {
20: public :
21: SDMComponent_ID source;          //data provider
22: SDMMessage_ID msg_id;           //unique id of data message as defined in xTEDS of data provider
23: char msg[BUFSIZE-29];          //buffer containing the raw data
24: short length;                   //length of msg
25: short seq_num;                  //a sequence number corresponding to seq_num from SDMSerreqst
26: SDMData();
27: long Send(const SDMComponent_ID& destination,long length);
28: long Send(const SDMComponent_ID& destination);
29: // Recv() is deprecated, to be removed in next major release
30: long Recv(long port,long length); SDM_DEPRECATED
31: long Marshal(char* buf,long length);
32: long Unmarshal(const char* buf,long length);
33: long Marshal(char* buf);
34: long Unmarshal(const char* buf);
35: void setSocket(int);
36: short getInt(long) const;
37: long getLong(long) const;
38: signed char getChar(long) const;
39: unsigned char getByte(long) const;
```

```
40: char* getString(long) const;
41: float getFloat(long) const;
42: double getDouble(long) const;
43: int MsgToString(char *str_buf, int length) const;
44: private :
45: long pid;           //SDM process id of xTEDS provider
46: };
47:
48: #endif
```

File: sdm/common/message/SDMCancelxTEDS.h

```
1: #ifndef __SDM_CANCEL_XTEDS_H_
2: #define __SDM_CANCEL_XTEDS_H_
3:
4: #include "SDMmessage.h"
5:
6: class SDMLIB_API SDMCancelxTEDS: public SDMmessage
7: {
8: public :
9:     SDMComponent_ID source;           //the xTEDS provider
10:    unsigned char fullCancel; //0 to deactivate an xTEDS, 1 to completely remove it
11:    SDMCancelxTEDS();
12:    long Send();
13:    long Marshal(char* buf);
14:    long Unmarshal(const char* buf);
15:    int MsgToString(char *str_buf, int length) const;
16: };
17:
18: #endif
```

File: sdm/common/message/SDMSubreqst.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMSubreqst.h"
4:
5: SDMSubreqst::SDMSubreqst():source(),destination(),msg_id(0,0),fault_id(0,0)
6: {
7:     MsgName = SDM_Subreqst;
8:     total_length = 24;
9: }
10:
11: long SDMSubreqst::Send()
12: {
13:     int result;
14:     result = SendTo(source);
15:     if (result <= 0)
16:         return SDM_MESSAGE_SEND_ERROR;
17:     return result;
18: }
19:
20: long SDMSubreqst::Recv(long port)
21: {
22:     int result;
23:     result = RecvFrom(port);
24:     if (result <= 0)
25:         return SDM_MESSAGE_RECV_ERROR;
26:     return result;
27: }
28:
29: long SDMSubreqst::Marshal(char* buf)
30: {
31:     int cur;
32:     cur = HEADER_SIZE;
33:     cur += source.Marshal(buf,cur);
34:     cur += destination.Marshal(buf,cur);
35:     cur += msg_id.Marshal(buf, cur);
36:     cur += fault_id.Marshal(buf, cur);
37:     msg_length = cur - HEADER_SIZE;
38:     MarshalHeader(buf);
39:     return cur;
```

```

40: }
41:
42: long SDMSubreqst::Unmarshal(const char* buf)
43: {
44:     int cur;
45:     cur = UnmarshalHeader(buf);
46:     if(cur == SDM_INVALID_MESSAGE)
47:     {
48:         return SDM_INVALID_MESSAGE;
49:     }
50:     if(total_length!=msg_length)
51:     {
52:         return SDM_INVALID_MESSAGE;
53:     }
54:     cur += source.Unmarshal(buf,cur);
55:     cur += destination.Unmarshal(buf,cur);
56:     cur += msg_id.Unmarshal(buf, cur);
57:     cur += fault_id.Unmarshal(buf, cur);
58: #ifdef BUILD_WITH_MESSAGE_LOGGING
59:     Logger.MessageReceived(*this);
60: #endif
61:     return cur;
62: }
63: int SDMSubreqst::MsgToString(char *str_buf, int length) const
64: {
65:     char source_id[40];
66:     char dest_id[40];
67:     char message_id[30];
68:     char flt_id[30];
69:
70:     if (length < 8096)
71:         return 0;
72:     source.IDToString(source_id, 40);
73:     destination.IDToString(dest_id, 40);
74:     msg_id.IDToString(message_id, 30);
75:     fault_id.IDToString(flt_id, 30);
76: #ifdef WIN32
77:     sprintf(str_buf,
78: #else
79:     snprintf(str_buf,length,
80: #endif

```

```
81:      "SDMSubreqst %ld:%ld %ld bytes, Source: %s Dest: %s msg_id: %s FaultID: %s", sec, subsec,  
msg_length, source_id, dest_id, message_id, flt_id);  
82: return (int)strlen(str_buf);  
83: }
```

File: sdm/common/message/SDMAck.h

```
1: #ifndef __SDM_ACK_H_
2: #define __SDM_ACK_H_
3: #include "SDMmessage.h"
4: #include "SDMComponent_ID.h"
5: #include "../SDMComHandle.h"
6:
7: //This class is intended for use by only SDM core components
8:
9: class SDMLIB_API SDMAck : public SDMmessage
10: {
11: public:
12: short error;      //Error code if any
13: SDMAck();
14: long Send();
15: long Send(const SDMComHandle& Handle);
16: long Marshal(char *buf);
17: long Unmarshal(const char *buf);
18: int MsgToString(char *buf, int length) const;
19: };
20:
21: #endif
```


File: sdm/common/message/SDMTat.h

```
1: #ifndef __SDM_TAT_H
2: #define __SDM_TAT_H
3:
4: #include "SDMmessage.h"
5:
6: class SDMLIB_API SDMTat: public SDMmessage
7: {
8: public :
9:     SDMComponent_ID destination;    //unique id of device
10:                                     //total time given as seconds + useconds
11:     unsigned long seconds;          //time in seconds
12:     unsigned long useconds;         //time in useconds
13:
14:     SDMTat();
15:     long Send();
16:     long Send(const SDMComponent_ID& destination);
17:     long Recv(long port); SDM_DEPRECATED
18:     long Marshal(char* buf);
19:     long Unmarshal(const char* buf);
20:     int MsgToString(char *str_buf, int length) const;
21: };
22:
23: #endif
```

File: sdm/common/message/SDMSearchReply.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMSearchReply.h"
4: #include "../UDPcom.h"
5:
6: SDMSearchReply::SDMSearchReply():source(),id(0),emptyflag(false)
7: {
8:     MsgName = SDM_SearchReply;
9:     captured_matches[0] = '\0';
10:    total_length = 13;
11: }
12:
13: long SDMSearchReply::Send(const SDMComponent_ID& destination)
14: {
15:     int result;
16:     emptyflag = false;
17:     result = SendTo(destination);
18:     if (result <= 0)
19:         return SDM_MESSAGE_SEND_ERROR;
20:     return result;
21: }
22:
23: long SDMSearchReply::Recv(long port)
24: {
25:     int result;
26:     result = RecvFrom(port);
27:     if (result <= 0)
28:         return SDM_MESSAGE_RECV_ERROR;
29:     if (result == 1)
30:         return SDM_NO_FURTHER_DATA_PROVIDER;
31:     return result;
32: }
33:
34: long SDMSearchReply::Marshal(char* buf)
35: {
36:     int cur;
37:     int bufmarker;
38:     if(emptyflag == true)
39: {
```

```

40:     cur = MarshalEmpty(buf);
41:     return cur;
42: }
43: cur = HEADER_SIZE;
44: cur += source.Marshal(buf,cur);
45: PUT_USHORT (&buf[cur], id);
46: cur += sizeof(id);
47: bufmarker = 0;
48: while(captured_matches[bufmarker] != 0)
49: {
50:     memcpy(buf+cur,captured_matches+bufmarker,strlen(captured_matches+bufmarker));
51:     cur += (int)strlen(captured_matches+bufmarker);
52:     buf[cur] = 0;
53:     cur++;
54:     bufmarker += (int)strlen(captured_matches+bufmarker);
55:     bufmarker++;
56: }
57: buf[cur] = 0;
58: cur++;
59: msg_length = cur - HEADER_SIZE;
60: MarshalHeader(buf);
61: return cur;
62: }
63:
64: long SDMSearchReply::Unmarshal(const char* buf)
65: {
66: int cur;
67: int bufmarker;
68: long captured_matches_msg_length;
69: cur = UnmarshalHeader(buf);
70: if(cur==SDM_INVALID_MESSAGE)
71: {
72:     return SDM_INVALID_MESSAGE;
73: }
74: if(msg_length == 0)
75:     return SDM_NO_FURTHER_DATA_PROVIDER;
76: if(total_length>msg_length)
77: {
78:     return SDM_INVALID_MESSAGE;
79: }
80: cur += source.Unmarshal(buf,cur);

```

```

81: id = GET_USHORT (&buf[cur]);
82: cur += sizeof(id);
83: bufmarker = 0;
84: while (buf[cur] != 0)
85: {
86:     captured_matches_msg_length = (long)strlen(buf + cur) + 1;
87:     memcpy(&captured_matches[bufmarker], buf + cur, captured_matches_msg_length);
88:     cur += captured_matches_msg_length;
89:     bufmarker += captured_matches_msg_length;
90: }
91: captured_matches[bufmarker++] = '\0';
92: cur++;
93: #ifdef BUILD_WITH_MESSAGE_LOGGING
94: Logger.MessageReceived(*this);
95: #endif
96: return cur;
97: }
98:
99: long SDMSearchReply::MarshalEmpty(char* buf)
100: {
101:     int cur;
102:     msg_length = 0;
103:     cur = MarshalHeader(buf);
104:     return cur;
105: }
106:
107: long SDMSearchReply::SendEmpty(const SDMComponent_ID& destination)
108: {
109:     int result;
110:     emptyflag = true;
111:     result = SendTo(destination);
112:     if (result <= 0)
113:         return SDM_MESSAGE_SEND_ERROR;
114:     return result;
115: }
116:
117: int SDMSearchReply::MsgToString(char *str_buf, int length) const
118: {
119:     char source_id[40];
120:     char MatchBuf[sizeof(captured_matches)];
121:

```

```

122:   if (length < 8096)
123:       return 0;
124:   source.IDToString(source_id, 40);
125:
126:   bool NullFound = false;
127:   for (unsigned int i = 0; i < sizeof(MatchBuf) - 1; i++)
128:   {
129:       if (captured_matches[i] == '\0')
130:       {
131:           MatchBuf[i] = '0';
132:           if (NullFound)
133:           {
134:               MatchBuf[++i] = '\0';
135:               break;
136:           }
137:           else
138:               NullFound = true;
139:       }
140:       else
141:       {
142:           NullFound = false;
143:           MatchBuf[i] = captured_matches[i];
144:       }
145:   }
146:
147: #ifdef WIN32
148:   sprintf(str_buf,
149: #else
150:   snprintf(str_buf, length,
151: #endif
152:           "SDMSearchReply %ld:%ld %ld bytes, Source: %s ID: %d captured_matches: \"%s\"", sec,
subsec, msg_length, source_id, id, MatchBuf);
153:   return (int)strlen(str_buf);
154: }

```

File: sdm/common/message/SDMDMLLeader.h

```
1: #ifndef __SDM_DM_LEADER_H_
2: #define __SDM_DM_LEADER_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMComponent_ID.h"
6:
7: //This class is intended for use by only SDM core components
8:
9: class SDMLIB_API SDMDMLLeader : public SDMmessage
10: {
11: public:
12:     SDMComponent_ID source;          //The DM who sent the message
13:     char running_flag;               //This flag specifies that this DM is already running and overrides any
                                     //smaller addresses
14:     SDMDMLLeader();
15:     ~SDMDMLLeader();
16:     long Marshal(char *buf);
17:     long Unmarshal(const char *buf);
18:     long Send();
19:     int MsgToString(char *str_buf, int length) const;
20: private:
21: };
22:
23: #endif
```

File: sdm/common/message/SDMError.h

```
1: #ifndef __SDM_ERROR_H
2: #define __SDM_ERROR_H
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: class SDMLIB_API SDMError: public SDMmessage
8: {
9: public :
10: SDMComponent_ID source;          //source of error
11: unsigned char error;             //an error code
12: SDMMessage_ID msg_id;           //message id producing the error
13: char error_msg[256];
14: SDMError();
15: long Send();
16: long Send(const SDMComponent_ID& destination);
17: long Recv(long port); SDM_DEPRECATED
18: long Marshal(char* buf);
19: long Unmarshal(const char* buf);
20: int MsgToString(char *str_buf, int length) const;
21: };
22:
23: #endif
```

File: sdm/common/message/SDMVarInfo.h

```
1: #ifndef __SDM_VAR_INFO_H_
2: #define __SDM_VAR_INFO_H_
3: #include "SDMmessage.h"
4: #include "SDMComponent_ID.h"
5: #include "SDMRegInfo.h"
6:
7: class SDMLIB_API SDMVarInfo : public SDMmessage
8: {
9: public:
10: SDMVarInfo();
11: SDMComponent_ID source;           //The source corresponding to the reply
12: unsigned short id;               //Identifier number of this request
13: char interface[XTEDS_MAX_ITEM_NAME_SIZE]; //The name of the Interface the variable
resides in
14: char var_xTEDS[BUFSIZE-78];      //Returned variable information
15: long Marshal(char *buf);
16: long Unmarshal(const char *buf);
17: long Send(const SDMComponent_ID& destination);
18: long Recv(long port); SDM_DEPRECATED
19: long MarshalEmpty(char* buf);
20: long SendEmpty(const SDMComponent_ID& destination);
21: int MsgToString(char *str_buf, int length) const;
22:
23: private:
24: bool emptyflag;
25: };
26:
27:
28:
29: #endif
```


File: sdm/common/message/SDMCancel.h

```
1: #ifndef __SDM_CANCEL_H_
2: #define __SDM_CANCEL_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: class SDMLIB_API SDMCancel: public SDMmessage
8: {
9: public :
10: SDMComponent_ID source;          //the data source
11: SDMComponent_ID destination;    //the subscriber
12: SDMMessage_ID msg_id;          //unique id of message to cancel
13: SDMCancel();
14: long Send();
15: long Marshal(char* buf);
16: long Unmarshal(const char* buf);
17: int MsgToString(char *buf, int length) const;
18: };
19:
20: #endif
```

File: sdm/common/message/SDMElection.h

```
1: #ifndef __SDM_ELECTION_H_
2: #define __SDM_ELECTION_H_
3:
4: #include "SDMmessage.h"
5:
6: //This class is intended for use by only SDM core components
7:
8: class SDMLIB_API SDMElection : public SDMmessage
9: {
10: public:
11: SDMElection();
12: ~SDMElection();
13: long Marshal(char *buf);
14: long Unmarshal(const char *buf);
15: int MsgToString(char *str_buf, int length) const;
16: long Send();
17: private:
18: };
19:
20: #endif
```

File: sdm/common/message/SDMxTEDS.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include <unistd.h>
4: #include "SDMxTEDS.h"
5: #include "../UDPcom.h"
6:
7: SDMxTEDS::SDMxTEDS():source(),active(1),pid(PID)
8: {
9:     MsgName = SDM_xTEDS;
10:    SPA_node[0] = '\0';
11:    xTEDS[0] = '\0';
12:    total_length = 17;
13: }
14:
15: long SDMxTEDS::Send()
16: {
17:     return SendDM();
18: }
19:
20: long SDMxTEDS::Recv(long port)
21: {
22:     int result;
23:     result = RecvFrom(port);
24:     if (result <= 0)
25:         return SDM_MESSAGE_RECV_ERROR;
26:     return result;
27: }
28:
29: long SDMxTEDS::Marshal(char* buf)
30: {
31:     int cur;
32:     unsigned int i,j;
33:
34:     cur = HEADER_SIZE;
35:
36:     cur += source.Marshal(buf,cur);
37:     PUT_CHAR(&buf[cur],active);
38:     cur += sizeof(active);
39:     PUT_LONG(&buf[cur], pid);
```

```

40: cur += sizeof(pid);
41:
42: //Start at byte two, the first byte could be zero if this is a segmented xTEDS
43: for(i=2; i<sizeof(xTEDS); i++)
44: {
45:     if(xTEDS[i] == '\0')
46:     {
47:         break; //find null terminator
48:     }
49: }
50: if (i >= sizeof(xTEDS) - 1)
51: {
52:     i = sizeof(xTEDS) - 2;
53: }
54: memcpy(buf+cur,xTEDS,i+1);
55: cur += i+1;
56: for(j=0; j<sizeof(SPA_node); j++)
57: {
58:     if(SPA_node[j] == '\0')
59:     {
60:         break; //find null terminator
61:     }
62: }
63: if (j < sizeof(SPA_node) - 1)
64: {
65:     j++;
66: }
67: memcpy(buf+cur,SPA_node,j);
68: cur += j;
69: msg_length = cur - HEADER_SIZE;
70: MarshalHeader(buf);
71: return cur;
72: }
73:
74: long SDMXTEDS::Unmarshal(const char* buf)
75: {
76: int cur;
77: unsigned int xteds_length,usb_length;
78: cur = UnmarshalHeader(buf);
79: if(cur==SDM_INVALID_MESSAGE)
80: {

```

```

81:     return SDM_INVALID_MESSAGE;
82: }
83: if(total_length>msg_length)
84: {
85:     return SDM_INVALID_MESSAGE;
86: }
87: cur += source.Unmarshal(buf,cur);
88: active = GET_CHAR (&buf[cur]);
89: cur += sizeof(active);
90: pid = GET_LONG (&buf[cur]);
91: cur += sizeof(pid);
92: //
93: // Determine whether this is a segmented xTEDS, or an empty xTEDS
94: bool IsSegmented = false;
95: if (buf[cur] <= MAX_XTEDS_SEQUENCE_VALUE && buf[cur+1] <=
MAX_XTEDS_SEQUENCE_VALUE+1 && buf[cur+2] > MAX_XTEDS_SEQUENCE_VALUE)
96:     IsSegmented = true;
97: //
98: // Pull out the xTEDS
99: if (IsSegmented) //If segmented, ignore the possible null in first two bytes
100:     xteds_length = strlen(buf+cur+2) + 2;
101: else //Otherwise, look at the first two bytes
102:     xteds_length = strlen(buf+cur);
103: //
104: // Check to make sure the xTEDS in the message doesn't go beyond the size of the buffer
105: if (xteds_length >= sizeof(xTEDS)-1)
106: {
107:     //If so, copy all that we can, and terminate the string
108:     strncpy(xTEDS, buf+cur, sizeof(xTEDS));
109:     xTEDS[sizeof(xTEDS)-1] = '\0';
110: }
111: else//Otherwise, copy out the string as usual
112:     memcpy(xTEDS,buf+cur,xteds_length+1);
113: //
114: // Pull out the USB path
115: cur += xteds_length+1;
116: usb_length = strlen(buf+cur);
117: //
118: // Check to make sure the USB path in the message doesn't go beyond the size of the buffer
119: if (usb_length >= sizeof(SPA_node)-1)
120: {

```

```

121:         //If it does, copy all we can and terminate the string
122:         strncpy(SPA_node, buf+cur, sizeof(SPA_node));
123:         SPA_node[sizeof(SPA_node)-1] = '\0';
124:     }
125:     else//Otherwise, copy out the string as usual
126:         memcpy(SPA_node,buf+cur,usb_length+1);
127:     cur += usb_length+1;
128: #ifdef BUILD_WITH_MESSAGE_LOGGING
129:     Logger.MessageReceived(*this);
130: #endif
131:     return cur;
132: }
133:
134: long SDMxTEDS::getPID(void)
135: {
136:     return pid;
137: }
138:
139: void SDMxTEDS::setPID(void)
140: {
141:     pid = PID;
142: }
143:
144: int SDMxTEDS::MsgToString(char *str_buf, int length) const
145: {
146:     char source_id[40];
147:
148:     if (length < 3*BUFSIZE)
149:         return 0;
150:     source.IDToString(source_id, 40);
151: #ifdef WIN32
152:     sprintf(str_buf,
153: #else
154:     snprintf(str_buf,3*BUFSIZE-1,
155: #endif
156:         "SDMxTEDS %ld:%ld %ld bytes, Source: %s PID: %ld Active: %d xTEDS: %s SPANode:
%s", sec, subsec, msg_length, source_id, pid, active, xTEDS, SPA_node);
157:     return strlen(str_buf);
158: }

```

File: sdm/common/message/SDMVarReq.h

```
1: #ifndef __SDM_VAR_REQ_H_
2: #define __SDM_VAR_REQ_H_
3: #include "SDMmessage.h"
4: #include "SDMComponent_ID.h"
5: #include "SDMMessage_ID.h"
6:
7: #define SDM_VARREQ_CURRENT          0
8: #define SDM_VARREQ_CURRENT_AND_FUTURE  5
9: #define SDM_VARREQ_CANCEL          11
10:
11: class SDMLIB_API SDMVarReq : public SDMmessage
12: {
13: public:
14: SDMVarReq();
15: SDMComponent_ID source;           //The id of the provider who's variables to request
16: SDMComponent_ID destination;      //The id of the requester
17: unsigned char reply;              //style of reply desired (see above)
18: SDMMMessage_ID msg_id;            //The interface containing the needed variables
19: unsigned short id;                //Identifier number of this request
20: char variable[XTEDS_MAX_ITEM_NAME_SIZE]; //The name of the variable
21: long Marshal(char *buf);
22: long Unmarshal(const char *buf);
23: long Send();
24: int MsgToString(char *str_buf, int length) const;
25: };
26:
27: #endif
```

File: sdm/common/message/SDMmessage.cpp

```
1: #include <stdlib.h>
2: #include <string.h>
3: #include <unistd.h>
4: #include "SDMmessage.h"
5: #include "../UDPcom.h"
6: #include "../TCPcom.h"
7: #include "../marshall.h"
8: #include <sys/time.h>
9: #include <sys/socket.h>
10: #include <netinet/in.h>
11: #include <arpa/inet.h>
12: #include <stdio.h>
13:
14: #ifndef WIN32
15: # include <net/if.h>
16: # include <netdb.h>
17: #endif
18: #include <sys/stat.h>
19:
20: #ifdef WIN32
21: #undef close
22: #define close closesocket
23: #endif
24:
25: SDMLIB_API SDMComponent_ID TaskManager;
26: SDMLIB_API SDMComponent_ID DataManager;
27: int PID;
28: #ifdef BUILD_WITH_MESSAGE_LOGGING
29: //Initialize the static member
30: SDMMessageLogger SDMmessage::Logger;
31: #endif
32:
33: SDMLIB_API void SDMInit(int argc, char** argv)
34: {
35: //initialize connection with SDM system
36: TaskManager.setPort(PORT_TM);
37: DataManager.setPort(PORT_DM);
38: if(argc == 1)
39: {
```



```

40:    //assume local local
41:    TaskManager.setAddress(inet_addr("127.0.0.1"));
42:    DataManager.setAddress(inet_addr("127.0.0.1"));
43:    PID = 0;
44: }
45: else
46: {
47:     if (strcmp(argv[1],"local")==0)
48:         TaskManager.setAddress(inet_addr("127.0.0.1"));
49:     else
50:         TaskManager.setAddress(inet_addr(argv[1]));
51:
52:     if (strcmp(argv[2],"local")==0)
53:         DataManager.setAddress(inet_addr("127.0.0.1"));
54:     else
55:         DataManager.setAddress(inet_addr(argv[2]));
56:     PID = atoi(argv[3]);
57: }
58: }
59:
60: SDMLIB_API long getPort()
61: {
62: char msg[5];
63: int sock;
64: long value = -1;
65:
66: msg[0] = SDM_ReqPort;
67: sock = UDPconnect("127.0.0.1", PORT_PM);
68: if (sock != IP_SOCKET_INVALID)
69: {
70:     UDPsend(sock, msg, 1);
71:     if(UDPrecv(sock, msg, 5) == 5)
72:     {
73:         value = GET_LONG (&msg[1]);
74:     }
75:     UDPclose(sock);
76: }
77: //If the Process Manager didn't respond
78: if (value < PORT_APP_START || value > 65535)
79: {
80:     return SDM_PM_NOT_AVAILABLE;

```

```

81: }
82:
83: return value;
84: }
85:
86: char SDMmessage::GetMsgName() const
87: {
88:     return MsgName;
89: }
90:
91: // This function is deprecated, to be removed in next major release
92: long SDMmessage::RecvFrom(long port)
93: {
94:     char buf[BUFSIZE];
95:     long result;
96:     long unmarshal_result;
97:     memset(buf,0,sizeof(buf));
98:     if(bound == IP_SOCK_INVALID)
99:     {
100:         bound = UDPpassive(port);
101:         if(bound == IP_SOCK_INVALID)
102:         {
103:             return bound;
104:         }
105:     }
106:     result = UDPServ_recv(bound,buf,BUFSIZE);
107:     if (result <= 0)
108:     {
109:         return result;
110:     }
111:     unmarshal_result = Unmarshal(buf);
112:     if (result < unmarshal_result)
113:     {
114:         return result;
115:     }
116:     return unmarshal_result;
117: }
118:
119:
120: long SDMmessage::SendDM()
121: {

```

```

122: #ifdef BUILD_FOR_PNPSAT
123:  struct hostent *he;
124:  unsigned long addr;
125:
126:  while ((he=gethostbyname("datamanager.spacewire")) == NULL)
127:  {
128:      sleep(1);
129:  }
130:  memcpy(&addr, he->h_addr, sizeof(addr));
131:
132:  DataManager.setAddress(addr);
133:  DataManager.setPort(PORT_DM);
134: #endif
135: #ifdef PNP_FAKE
136: //  struct hostent *he;
137: //  unsigned long addr;
138: //
139: //  while ((he=gethostbyname("datamanager")) == NULL)
140: //  {
141: //      sleep(1);
142: //  }
143: //  memcpy(&addr, he->h_addr, sizeof(addr));
144: //
145: //  DataManager.setAddress(addr);
146: //  DataManager.setPort(PORT_DM);
147: #endif
148:  return SendTo(DataManager);
149: }
150:
151: long SDMmessage::SendTM()
152: {
153:  return SendTo(TaskManager);
154: }
155:
156: long SDMmessage::SendTo(const SDMComponent_ID& destination)
157: {
158:  int sock;
159:  long i;
160:  long result;
161:  char buf[LARGE_MSG_BUFSIZE];
162:  char ack[16];

```

```

163: long address = destination.getAddress();
164: int timeout = 2000; // 2 sec
165: int count = 0;
166: short error = 0;
167:
168: //fill buffer
169: i = Marshal(buf);
170: if(i < 0)
171:     return SDM_MESSAGE_SEND_ERROR;
172: #ifdef TCP_TRANSMIT_OF_LARGE_XTEDS
173:     if(i > BUFSIZE)
174:         return SendTCP(destination.getAddress(), destination.getPort());
175: #endif
176: //send message
177: sock = UDPconnect(address, destination.getPort());
178: if (sock != IP_SOCK_INVALID) // cleanup issue 26
179: {
180:     result = UDPsend(sock, buf, i);
181:     //Look for SDM_ACK from DM for certain messages
182:     if(buf[0] == SDM_xTEDS || buf[0] == SDM_CancelxTEDS)
183:     {
184:         if(buf[0] == SDM_xTEDS)
185:         {
186:             timeout = 5000; // 5 sec
187:         }
188:         UDPset_rcv_timeout(sock, timeout);
189:
190:         UDPrcv(sock, &ack, 13);
191:         while(ack[0] != SDM_ACK && count < NUMRETRIES)
192:         {
193:             count++;
194:
195:             result = UDPsend(sock, buf, i);
196:             UDPrcv(sock, &ack, 13);
197:         }
198:         //remove timeout on sdm_dm_sock
199:         UDPset_rcv_timeout(sock, 0);
200:
201:         if(count == NUMRETRIES && ack[0] != SDM_ACK)
202:         {
203:             result = SDM_UNABLE_TO_REGISTER;

```

```

204:     }
205:     else
206:     {
207:         error = GET_SHORT(&ack[11]);
208:         if(error < 0)
209:         {
210:             result = error;
211:         }
212:     }
213: }
214:     UDPclose (sock);
215: #ifdef BUILD_WITH_MESSAGE_LOGGING
216:     Logger.MessageSent(*this);
217: #endif
218: }
219:     return result;
220: }
221:
222: /*
223:  Forward a previously received message unchanged (i.e. without changing the timestamp).
224:
225:  Implementation notes: Currently, only messages of maximum size BUFSIZE can be forwarded.
226:  The only message that won't send as expected is SDMxTEDS. The reason for this is that
227:  the DM is the only application that should ever receive SDMxTEDS, and it will never use
228:  this function.
229:
230:  Params:
231:  destination - The component id to which to forward the message
232:  Returns:
233:  long - The length of the message sent or -1 to indicate failure
234:  */
235: long SDMmessage::Forward(const SDMComponent_ID& destination)
236: {
237:     char buf[BUFSIZE];
238:
239:     if (MsgName == SDM_xTEDS)
240:         return -1;
241:
242:     // Save the old timestamp
243:     long OldSeconds = sec;
244:     long OldSubSeconds = subsec;

```

```

245:
246: // Marshal the message, this call will assign a new timestamp
247: const long MessageLength = Marshal(buf);
248: if (MessageLength < 0)
249:     return MessageLength;
250:
251: // Reset the old timestamp
252: sec = OldSeconds;
253: subsec = OldSubSeconds;
254:
255: // Remarshal the header, overwriting previous data, body of message is still intact
256: MarshalHeaderOldTimeStamp(buf);
257:
258: // Now send the message to the destination component id
259: int sock = UDPconnect(destination.getAddress(), destination.getPort());
260: if (sock < 0)
261:     return -1;
262: long SendResult = UDPsend(sock, buf, MessageLength);
263: UDPclose(sock);
264:
265: #ifdef BUILD_WITH_MESSAGE_LOGGING
266:     Logger.MessageSent(*this);
267: #endif
268:     return SendResult;
269: }
270:
271: long SDMmessage::SendTCP(long ip_addr,long port)
272: {
273:     int sock;
274:     long i;
275:     long result;
276:     char buf[3*BUFSIZE];
277:     char ack[16];
278:     short error = 0;
279:
280:     //fill buffer
281:     i = Marshal(buf);
282:     if(i < 0)
283:         return SDM_MESSAGE_SEND_ERROR;
284:     //send message
285:     sock = TCPconnect(ip_addr,port);

```

```

286: if (sock != IP_SOCK_INVALID)
287: {
288:     result = TCPsend(sock,buf,i);
289:     if(buf[0] == SDM_xTEDS || buf[0] == SDM_CancelxTEDS)
290:     {
291:         TCPrecv(sock,ack,13);
292:         error = GET_SHORT(&ack[HEADER_SIZE]);
293:         if(error < 0)
294:         {
295:             result = error;
296:         }
297:     }
298:     TCPclose (sock);
299: #ifdef BUILD_WITH_MESSAGE_LOGGING
300:     Logger.MessageSent(*this);
301: #endif
302: }
303: return result;
304: }
305:
306: //If addr is supplied, it must be in network byte order
307: long SDMmessage::SendBroadcast(long address, unsigned short dest_port)
308: {
309:     int i, result;
310:     char buf[BUFSIZE];
311:     long b_cast_address;
312:     if (address == 0)          //If no address was supplied
313:         b_cast_address = DataManager.getAddress();
314:     else
315:         b_cast_address = address;
316:
317:     // DANGEROUS: assumes a little-endian machine and a /24 subnet!
318:     b_cast_address |= 0xFF000000;    //Set the broadcast address's last octet to 255 as
xxx.xxx.xxx.255
319:     i = Marshal(buf);
320:     if (i < 0)
321:         return SDM_MESSAGE_SEND_ERROR;
322:
323:     result = UDPsend_broadcast(b_cast_address, dest_port, buf, i);
324:     if (result < 0)
325:         return SDM_MESSAGE_SEND_ERROR;

```

```

326: #ifdef BUILD_WITH_MESSAGE_LOGGING
327:     Logger.MessageSent(*this);
328: #endif
329:     return i;
330: }
331:
332: long SDMmessage::SendReplyTo(int socket, const struct sockaddr_in* sin, bool tcp)
333: {
334:     long i;
335:     long result;
336:     char buf[BUFSIZE];
337:
338:     if(socket == IP_SOCKET_INVALID) //This is a bad socket descriptor and no message should be
sent
339:         return 0;
340:
341:     i = Marshal(buf);
342:     if (i < 0)
343:         return SDM_MESSAGE_SEND_ERROR;
344:
345:     if(tcp)
346:         result = TCPserv_replyto(socket,buf,i,sin);
347:     else
348:         result = UDPServ_replyto(socket,buf,i,sin);
349:
350:     if(result < 0)
351:     {
352: #ifndef WIN32
353:         perror("Unable to send reply message: ");
354:     #else
355:         printf("Unable to send reply message: %d \n",WSAGetLastError());
356:     #endif
357:
358:         return SDM_MESSAGE_SEND_ERROR;
359:     }
360: #ifdef BUILD_WITH_MESSAGE_LOGGING
361:     Logger.MessageSent(*this);
362: #endif
363:     return i;
364: }
365:

```



```

366: /*
367:    Marshal the header without reassigning a new timestamp.
368: */
369: long SDMmessage::MarshalHeaderOldTimeStamp(char* buf)
370: {
371:     buf[0] = MsgName;
372:     PUT_LONG(&buf[1],sec);
373:     PUT_LONG(&buf[5],subsec);
374:     PUT_USHORT(&buf[9],msg_length);
375:     return HEADER_SIZE;
376: }
377:
378: long SDMmessage::MarshalHeader(char* buf)
379: {
380: #ifdef __VXWORKS__
381:     struct timespec time;
382:     clock_gettime(CLOCK_REALTIME, &time);
383:     sec = time.tv_sec;
384:     subsec = time.tv_nsec/1000;
385: #else
386:     struct timeval time;
387:     gettimeofday(&time,NULL);
388:     sec = time.tv_sec;
389:     subsec = time.tv_usec;
390: #endif
391:     buf[0] = MsgName;
392:     PUT_LONG(&buf[1],sec);
393:     PUT_LONG(&buf[5],subsec);
394:     PUT_USHORT(&buf[9],msg_length);
395:     return HEADER_SIZE;
396: }
397:
398: long SDMmessage::UnmarshalHeader(const char* buf)
399: {
400:     if(buf == NULL || buf[0] != MsgName)
401:     {
402:         return SDM_INVALID_MESSAGE;
403:     }
404:     sec = GET_LONG(&buf[1]);
405:     subsec = GET_LONG(&buf[5]);
406:     msg_length = GET_USHORT(&buf[9]);

```

```
407:    return HEADER_SIZE;
408: }
```

File: sdm/common/message/SDMSubreqst.h

```
1: #ifndef __SDM_SUBREQST_H_
2: #define __SDM_SUBREQST_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: class SDMLIB_API SDMSubreqst: public SDMmessage
8: {
9: public :
10: SDMComponent_ID source;          //the data source
11: SDMComponent_ID destination;    //the subscriber
12: SDMMessage_ID msg_id;           //the message id of the message wanted
13: SDMMessage_ID fault_id;         //unique id of possible fault message, may be 0 for no fault
14: SDMSubreqst();
15: long Send();
16: long Recv(long port); SDM_DEPRECATED
17: long Marshal(char* buf);
18: long Unmarshal(const char* buf);
19: int MsgToString(char *str_buf, int length) const;
20: };
21:
22: #endif
```

File: sdm/common/message/SDMTaskError.h

```
1: #ifndef __SDM_TASK_ERROR_H_
2: #define __SDM_TASK_ERROR_H_
3:
4: #include "SDMmessage.h"
5:
6: //This class is intended only for use by SDM core components
7:
8: class SDMLIB_API SDMTaskError : public SDMmessage
9: {
10: public:
11:     SDMComponent_ID source;           //The PM on which the finished task resides
12:     int status;                       //The status of the error
13:     char filename[MAX_FILENAME_SIZE]; //The name of the task
14:     unsigned int pid;
15:     SDMTaskError();
16:     long Send();
17:     long Marshal (char* buf);
18:     long Unmarshal (const char* buf);
19:     int MsgToString(char *str_buf, int length) const;
20: };
21:
22:
23: #endif
```

File: sdm/common/message/SDMSearchReply.h

```
1: #ifndef __SDM_SEARCHREPLY_H_
2: #define __SDM_SEARCHREPLY_H_
3:
4: #include "SDMmessage.h"
5:
6: class SDMLIB_API SDMSearchReply: public SDMmessage
7: {
8: public :
9:     SDMComponent_ID source;           //unique id of xTEDS provider
10:    unsigned short id;                 //the id of the SDMReqReg resulting in this SDMSearchReply
11:    char captured_matches[BUFSIZE-23]; //the captured matches from the search
12:    SDMSearchReply();
13:    long Send(const SDMComponent_ID& destination);
14:    long Recv(long port); SDM_DEPRECATED
15:    long Marshal(char* buf);
16:    long Unmarshal(const char* buf);
17:    long MarshalEmpty(char* buf);
18:    long SendEmpty(const SDMComponent_ID& destination);
19:    int MsgToString(char *buf, int length) const;
20:
21: private:
22:    bool emptyflag;                   //This flag is set for sending and empty SDMSearchReply message
23: };
24:
25: #endif
26:
```

File: sdm/common/message/SDMRegister.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "SDMRegister.h"
4:
5: SDMRegister::SDMRegister()
6: {
7:     MsgName = SDM_Register;
8:     total_length = 2;
9:     sensorIndex = -1;
10: }
11:
12: long SDMRegister::Marshal(char *buf)
13: {
14:     int cur = HEADER_SIZE;
15:     PUT_SHORT(&buf[cur], sensorIndex);
16:     cur += sizeof(short);
17:     msg_length = cur - HEADER_SIZE;
18:     MarshalHeader(buf);
19:     return cur;
20: }
21:
22: long SDMRegister::Unmarshal(const char*buf)
23: {
24:     int cur = UnmarshalHeader(buf);
25:     if(cur == SDM_INVALID_MESSAGE || total_length != msg_length)
26:     {
27:         return SDM_INVALID_MESSAGE;
28:     }
29:     sensorIndex = GET_SHORT(&buf[cur]);
30:     cur += sizeof(short);
31: #ifdef BUILD_WITH_MESSAGE_LOGGING
32:     Logger.MessageReceived(*this);
33: #endif
34:     return cur;
35: }
36:
37: int SDMRegister::MsgToString(char *str_buf, int length) const
38: {
39:     if (length < 8096)
```

```
40:     return 0;
41: #ifdef WIN32
42:     sprintf(str_buf,
43: #else
44:     snprintf(str_buf,length,
45: #endif
46:     "SDMRegister %ld:%ld %ld bytes", sec, subsec, msg_length);
47:     return (int)strlen(str_buf);
48: }
```

File: sdm/common/message/SDMRegInfo.h

```
1: #ifndef __SDM_REGINFO_H_
2: #define __SDM_REGINFO_H_
3:
4: #define SDM_REGINFO_REGISTRATION 0
5: #define SDM_REGINFO_CANCELLATION 1
6:
7: #include "SDMmessage.h"
8: #include "SDMMessage_ID.h"
9:
10: class SDMLIB_API SDMRegInfo: public SDMmessage
11: {
12: public :
13: unsigned short id;           //the id of the SDMReqReg resulting in this SDMRegInfo
14: unsigned char type;         //switch indicating if message is the result of a registration or
cancellation
15: SDMComponent_ID source;      //unique id of xTEDS provider
16: SDMMessage_ID msg_id;        //unique id of data message
17: char msg_def[MSG_DEF_SIZE];  //annotated summary of data message
18: char xTEDS_section[MSG_DEF_SIZE]; //section of the full data message definition in the
xTEDS
19: SDMRegInfo();
20: long Send(const SDMComponent_ID& destination);
21: long Recv(long port); SDM_DEPRECATED
22: long Marshal(char* buf);
23: long Unmarshal(const char* buf);
24: long MarshalEmpty(char* buf);
25: long SendEmpty(const SDMComponent_ID& destination);
26: int MsgToString(char *str_buf, int length) const;
27:
28: private:
29: bool emptyflag;             //This flag is set for sending and empty SDMRegInfo message
30: };
31:
32: #endif
```


File: sdm/common/message/SDMMessage_ID.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMMessage_ID.h"
4: #include "../marshall.h"
5:
6: SDMMessage_ID::SDMMessage_ID() : _interface(0), _message(0)
7: {
8: }
9:
10: SDMMessage_ID::SDMMessage_ID(unsigned char interface_id, unsigned char msg_id) :
    _interface(interface_id), _message(msg_id)
11: {
12: }
13:
14: SDMMessage_ID::SDMMessage_ID(int interface_id, int msg_id) : _interface(0), _message(0)
15: {
16: setInterface(interface_id);
17: setMessage(msg_id);
18: }
19:
20: void SDMMessage_ID::setInterface(int interface_num)
21: {
22: //Mask off all but lowest byte
23: unsigned char c_value = static_cast<unsigned char> (interface_num&0xFF);
24: setInterface(c_value);
25: }
26:
27: void SDMMessage_ID::setMessage(int msg_num)
28: {
29: //Mask off all but lowest byte
30: unsigned char c_value = static_cast<unsigned char> (msg_num&0xFF);
31: setMessage(c_value);
32: }
33:
34: short SDMMessage_ID::getInterfaceMessagePair() const
35: {
36: return (short)(_message + (_interface << 8));
37: }
38: bool SDMMessage_ID::operator== (const SDMMessage_ID &other) const
```

```

39: {
40: if (this->_message == other._message)
41:     if (this->_interface == other._interface)
42:         return true;
43: return false;
44: }
45: bool SDMMMessage_ID::operator== (const short &other) const
46: {
47: if (_interface == (other >> 8))
48:     if (_message == (other&0x00FF))
49:         return true;
50: return false;
51: }
52: SDMMMessage_ID& SDMMMessage_ID::operator= (const SDMMMessage_ID &other)
53: {
54: _interface = other._interface;
55: _message = other._message;
56: return *this;
57: }
58: SDMMMessage_ID& SDMMMessage_ID::operator= (const int value)
59: {
60: _interface = (0x0000ff00&value) >> 8;
61: _message = (0x000000ff&value);
62: return *this;
63: }
64:
65: int SDMMMessage_ID::Marshal(char *buf, int start)
66: {
67: int cur = 0;
68: PUT_UCHAR(&buf[start], _interface);
69: PUT_UCHAR(&buf[start+1], _message);
70: cur +=2;
71: return cur;
72: }
73:
74: int SDMMMessage_ID::Unmarshal(const char *buf, int start)
75: {
76: int cur = 0;
77: _interface = GET_UCHAR(&buf[start]);
78: _message = GET_UCHAR(&buf[start+1]);
79: cur +=2;

```

```
80: return cur;
81: }
82:
83: int SDMMMessage_ID::IDToString(char *buf, int length) const
84: {
85:     if (length < 30)
86:         return 0;
87: #ifdef WIN32
88:     sprintf(buf, "Interface: %d Message: %d", getInterface(), getMessage());
89: #else
90:     snprintf(buf, length, "Interface: %d Message: %d", getInterface(), getMessage());
91: #endif
92:     return (int)strlen(buf);
93: }
```

File: sdm/common/message/SDMSerreqst.h

```
1: #ifndef __SDM_SERREQUEST_H_
2: #define __SDM_SERREQUEST_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: class SDMLIB_API SDMSerreqst: public SDMmessage
8: {
9: public :
10: SDMComponent_ID source;          //the service provider
11: SDMComponent_ID destination;     //the subscriber
12: SDMMessage_ID command_id; //unique id of command portion of service
13: SDMMessage_ID reply_id;         //unique id of expected data reply message
14: SDMMessage_ID fault_id;         //unique id of possible fault message, may be 0 for no fault
15: short length;                  //length of data portion
16: short seq_num;                 //a sequence number for sync purposes
17: char data[BUFSIZE-39];         //raw data to be processed
18: SDMSerreqst();
19: long Send(const SDMComponent_ID& destination);
20: long Recv(long port); SDM_DEPRECATED
21: long Marshal(char* buf);
22: long Unmarshal(const char* buf);
23: int MsgToString(char *str_buf, int length) const;
24: };
25:
26: #endif
```

File: sdm/common/message/SDMHello.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include "SDMHello.h"
4:
5: SDMHello::SDMHello() : type('D')
6: {
7:     MsgName = SDM_Hello;
8:     total_length = 11;
9: }
10:
11: long SDMHello::Send()
12: {
13:     return SendDM();
14: }
15:
16: long SDMHello::Marshal(char* buf)
17: {
18:     int cur = HEADER_SIZE;
19:     cur += source.Marshal(buf,cur);
20:     PUT_CHAR(&buf[cur], type);
21:     cur += sizeof(char);
22:     msg_length = cur - HEADER_SIZE;
23:     MarshalHeader(buf);
24:     return cur;
25: }
26:
27: long SDMHello::Unmarshal(const char* buf)
28: {
29:     int cur = UnmarshalHeader(buf);
30:     if(cur == SDM_INVALID_MESSAGE || total_length != msg_length)
31:     {
32:         return SDM_INVALID_MESSAGE;
33:     }
34:     cur += source.Unmarshal(buf,cur);
35:     type = GET_CHAR(&buf[cur]);
36:     cur += sizeof(char);
37: #ifdef BUILD_WITH_MESSAGE_LOGGING
38:     Logger.MessageReceived(*this);
39: #endif
```

```

40: return cur;
41: }
42:
43: int SDMHHello::MsgToString(char* buf, int length) const
44: {
45:     if (length < 8096)
46:         return 0;
47: #ifdef WIN32
48:     sprintf(buf,
49: #else
50:     snprintf(buf,length,
51: #endif
52:     "SDMHHello %ld:%ld %ld bytes, error: %c", sec, subsec, msg_length, type);
53:     return (int)strlen(buf);
54: }

```

File: sdm/common/message/SDMVarInfo.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMVarInfo.h"
4:
5: SDMVarInfo::SDMVarInfo():source(),id(0),emptyflag(false)
6: {
7:     MsgName = SDM_VarInfo;
8:     total_length = 14;
9:     interface[0] = '\0';
10:    var_xTEDS[0] = '\0';
11: }
12:
13: long SDMVarInfo::Marshal(char *buf)
14: {
15:     int cur = HEADER_SIZE;
16:     if(emptyflag == true)
17:     {
18:         cur = MarshalEmpty(buf);
19:         return cur;
20:     }
21:     cur += source.Marshal(buf, cur);
22:     PUT_USHORT(buf+cur, id);
23:     cur += sizeof(id);
24:     strcpy(buf+cur, interface); //Copy the interface string field
25:     cur += strlen(interface) + 1; //Cur pointer is the size of the string plus null terminator
26:     strcpy(buf+cur, var_xTEDS); //Copy the variable string field
27:     cur += strlen(var_xTEDS) + 1; //Cur pointer is the size of the string plus null terminator
28:     msg_length = cur - HEADER_SIZE;
29:     MarshalHeader(buf);
30:     return cur;
31: }
32:
33: long SDMVarInfo::Unmarshal(const char *buf)
34: {
35:     int cur = UnmarshalHeader(buf);
36:     if (cur == SDM_INVALID_MESSAGE)
37:         return SDM_INVALID_MESSAGE;
38:     if (msg_length == 0)
39:         return SDM_NO_FURTHER_DATA_PROVIDER;
```

```

40: if (total_length > msg_length)
41:     return SDM_INVALID_MESSAGE;
42: cur += source.Unmarshal(buf, cur);
43: id = GET_USHORT(buf+cur);
44: cur += sizeof(id);
45: strcpy(interface, buf+cur);
46: cur += strlen(interface) + 1;
47: strcpy(var_xTEDS, buf+cur);
48: cur += strlen(var_xTEDS) + 1;
49: #ifdef BUILD_WITH_MESSAGE_LOGGING
50: Logger.MessageReceived(*this);
51: #endif
52: return cur;
53: }
54:
55: long SDMVarInfo::MarshalEmpty(char* buf)
56: {
57: int cur;
58: msg_length = 0;
59: cur = MarshalHeader(buf);
60: return cur;
61: }
62:
63: long SDMVarInfo::Send(const SDMComponent_ID& destination)
64: {
65: int result;
66: emptyflag = false;
67: result = SendTo(destination);
68: if (result <= 0)
69:     return SDM_MESSAGE_SEND_ERROR;
70: return result;
71: }
72:
73: long SDMVarInfo::Recv(long port)
74: {
75: int result;
76: result = RecvFrom(port);
77: if (result <= 0)
78:     return SDM_MESSAGE_RECV_ERROR;
79: if (result == 1)
80:     return SDM_NO_FURTHER_DATA_PROVIDER;

```



```

81: return result;
82: }
83:
84: long SDMVarInfo::SendEmpty(const SDMComponent_ID& destination)
85: {
86: int result;
87: emptyflag = true;
88: result = SendTo(destination);
89: if (result <= 0)
90:     return SDM_MESSAGE_SEND_ERROR;
91: return result;
92: }
93:
94: int SDMVarInfo::MsgToString(char *str_buf, int length) const
95: {
96: char source_id[40];
97:
98: if (length < 8096)
99:     return 0;
100: source.IDToString(source_id, 40);
101: #ifdef WIN32
102:     sprintf(str_buf,
103: #else
104:     snprintf(str_buf,length,
105: #endif
106:         "SDMVarInfo %ld:%ld %ld bytes, Source: %s id: %hu interface: %s var_xTEDS: %s", sec,
subsec, msg_length, source_id, id, interface, var_xTEDS);
107:     return strlen(str_buf);
108: }

```

File: sdm/common/message/SDMCommand.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMCommand.h"
4:
5: SDMCommand::SDMCommand():fault_id(0,0)
6: {
7:     MsgName = SDM_Command;
8:     total_length = 24;
9: }
10:
11: long SDMCommand::Send()
12: {
13:     return SendDM();
14: }
15:
16: long SDMCommand::Send(const SDMComponent_ID& destination)
17: {
18:     return SendTo(destination);
19: }
20:
21: long SDMCommand::Recv(long port)
22: {
23:     int result;
24:     result = RecvFrom(port);
25:     if (result <= 0)
26:         return SDM_MESSAGE_RECV_ERROR;
27:     return result;
28: }
29:
30: long SDMCommand::Marshal(char* buf)
31: {
32:     int cur = HEADER_SIZE;
33:     cur += source.Marshal(buf,cur);
34:     cur += destination.Marshal(buf,cur);
35:     cur += command_id.Marshal(buf, cur);
36:     cur += fault_id.Marshal(buf, cur);
37:     memcpy(buf+cur,&data,length);
38:     cur+=length;
39:     msg_length = cur - HEADER_SIZE;
```

```

40: MarshalHeader(buf);
41: return cur;
42: }
43:
44: long SDMCommand::Unmarshal(const char* buf)
45: {
46: int cur;
47: cur = UnmarshalHeader(buf);
48: if(cur==SDM_INVALID_MESSAGE)
49: {
50:     return SDM_INVALID_MESSAGE;
51: }
52: if(total_length>msg_length)
53: {
54:     return SDM_INVALID_MESSAGE;
55: }
56: cur += source.Unmarshal(buf,cur);
57: cur += destination.Unmarshal(buf,cur);
58: cur += command_id.Unmarshal(buf, cur);
59: cur += fault_id.Unmarshal(buf, cur);
60: length = msg_length - cur + HEADER_SIZE;
61: memcpy(&data,buf+cur,length);
62: #ifdef BUILD_WITH_MESSAGE_LOGGING
63: Logger.MessageReceived(*this);
64: #endif
65: return msg_length+HEADER_SIZE;
66: }
67: int SDMCommand::MsgToString(char *str_buf, int length) const
68: {
69: char source_id[40];
70: char destination_id[40];
71: char data_section[511];
72: char cmd_id[30];
73: char flt_id[30];
74: int i, j;
75: if (length < 8096)
76:     return 0;
77: for (i = 0, j = 0; j < this->length && i < 510; i+=2, j++)
78: {
79:     sprintf(&data_section[i], "%.2x", data[j]);
80: }

```

```

81: data_section[i] = '\0';
82:
83: source.IDToString(source_id, 40);
84: destination.IDToString(destination_id, 40);
85: command_id.IDToString(cmd_id, 30);
86: fault_id.IDToString(flt_id, 30);
87: #ifdef WIN32
88: sprintf(str_buf,
89: #else
90: snprintf(str_buf,length,
91: #endif
92:      "SDMCommand %ld:%ld %ld bytes, Source: %s Destination: %s CommandID: %s FaultID: %s
[DATA SECTION: %s ]", sec, subsec, msg_length, source_id, destination_id, cmd_id, flt_id,
data_section);
93: return (int)strlen(str_buf);
94: }

```

File: sdm/common/message/SDMHeartbeat.h

```
1: #ifndef __SDM_HEARTBEAT_H_
2: #define __SDM_HEARTBEAT_H_
3:
4: #include "SDMmessage.h"
5:
6: extern SDMLIB_API void SendHeartbeat();
7:
8: class SDMLIB_API SDMHeartbeat: public SDMmessage
9: {
10: public:
11:     SDMComponent_ID source;           //The source of the heartbeat message
12:     SDMHeartbeat();
13:     long Marshal(char *buf);
14:     long Unmarshal(const char * buf);
15:     int MsgToString(char *str_buf, int length) const;
16: };
17: #endif
```

File: sdm/common/message/SDMReqCode.h

```
1: #ifndef __SDM_REQCODE_H_
2: #define __SDM_REQCODE_H_
3:
4: #include "SDMmessage.h"
5:
6: //This class is intended only for use by SDM core components
7:
8: class SDMLIB_API SDMReqCode : public SDMmessage
9: {
10: public:
11: char filename[MAX_FILENAME_SIZE]; //The name of the file wanted
12: SDMComponent_ID source;           //The originator of the message
13: unsigned short seq_num;           //The sequence from which code should be transferred
14: int version;                      //Version of the code to request
15: SDMReqCode();
16: long Send();
17: long Recv(long port); SDM_DEPRECATED
18: long Marshal(char* buf);
19: long Unmarshal(const char* buf);
20: int MsgToString(char *str_buf, int length) const;
21: };
22:
23: #endif
```

File: sdm/common/message/SDMPostTask.h

```
1: #ifndef __SDM_POSTTASK_H_
2: #define __SDM_POSTTASK_H_
3:
4: #include "SDMmessage.h"
5:
6:
7: class SDMLIB_API SDMPostTask: public SDMmessage
8: {
9: public :
10: unsigned short resources;      //resource requirements of task
11: char priority;                //meaning of priority is TBD
12: int sched_interval;           //the time the TM waits until it posts the task
13: int mode;
14: int version;                  //specify a version number or location to run
15: char filename[MAX_FILENAME_SIZE]; //filename of task to be executed
16: SDMPostTask();
17: long Send();
18: long Marshal(char* buf);
19: long Unmarshal(const char* buf);
20: int MsgToString(char *str_buf, int length) const;
21: };
22:
23: #endif
```

File: sdm/common/message/SDMID.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMID.h"
4:
5:
6: SDMID::SDMID():destination()
7: {
8:     MsgName = SDM_ID;
9:     total_length = 10;
10: }
11:
12: long SDMID::Marshal(char *buf)
13: {
14:     int cur = HEADER_SIZE;
15:     cur += destination.Marshal(buf, cur);
16:     msg_length = cur - HEADER_SIZE;
17:     MarshalHeader(buf);
18:     return cur;
19: }
20:
21: long SDMID::Unmarshal(const char * buf)
22: {
23:     int cur = UnmarshalHeader(buf);
24:     if (cur==SDM_INVALID_MESSAGE)
25:     {
26:         return SDM_INVALID_MESSAGE;
27:     }
28:     if (total_length != msg_length)
29:     {
30:         return SDM_INVALID_MESSAGE;
31:     }
32:     cur += destination.Unmarshal(buf, cur);
33: #ifdef BUILD_WITH_MESSAGE_LOGGING
34:     Logger.MessageReceived(*this);
35: #endif
36:     return cur;
37: }
38:
39: int SDMID::MsgToString(char * str_buf, int length) const
```



```
40: {
41: char destination_id[40];
42:
43: if (length < 8096)
44:     return 0;
45: destination.IDToString(destination_id, 40);
46: #ifdef WIN32
47: sprintf(str_buf,
48: #else
49: snprintf(str_buf,length,
50: #endif
51:     "SDMID %ld:%ld %ld bytes, Source: %s", sec, subsec, msg_length, destination_id);
52: return (int)strlen(str_buf);
53: }
```

File: sdm/common/message/SDMService.h

```
1: #ifndef __SDM_SERVICE_H_
2: #define __SDM_SERVICE_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: class SDMLIB_API SDMService: public SDMmessage
8: {
9: public :
10: SDMComponent_ID source;          //the service provider
11: SDMComponent_ID destination;     //the subscriber
12: SDMMessage_ID command_id; //unique id of command portion of service
13: short length;                   //number of bytes in the data portion
14: short seq_num;                  //a sequence number for sync purposes
15: char data[BUFSIZE-35];          //data to be passed to service provider
16: SDMService();
17: long Send();
18: long Recv(long port); SDM_DEPRECATED
19: long Marshal(char* buf);
20: long Unmarshal(const char* buf);
21: int MsgToString(char *str_buf, int length) const;
22: };
23:
24: #endif
```

File: sdm/common/message/SDMTat.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMTat.h"
4: #include "../UDPcom.h"
5:
6: SDMTat::SDMTat():destination(),seconds(0),useconds(0)
7: {
8:     MsgName = SDM_Tat;
9:     total_length = 18;
10: }
11:
12: long SDMTat::Send()
13: {
14:     return SendDM();
15: }
16:
17: long SDMTat::Recv(long port)
18: {
19:     long result;
20:     result = RecvFrom(port);
21:     if (result <= 0)
22:         return SDM_MESSAGE_RECV_ERROR;
23:     return result;
24: }
25:
26: long SDMTat::Marshal(char* buf)
27: {
28:     int cur;
29:     cur = HEADER_SIZE;
30:     cur += destination.Marshal(buf,cur);
31:     PUT_ULONG (&buf[cur], seconds);
32:     cur += sizeof(seconds);
33:     PUT_ULONG (&buf[cur], useconds);
34:     cur += sizeof(useconds);
35:     msg_length = cur - HEADER_SIZE;
36:     MarshalHeader(buf);
37:     return cur;
38: }
39:
```

```

40: long SDMTat::Unmarshal(const char* buf)
41: {
42:     int cur;
43:     cur = UnmarshalHeader(buf);
44:     if(cur == SDM_INVALID_MESSAGE)
45:     {
46:         return SDM_INVALID_MESSAGE;
47:     }
48:     if(total_length!=msg_length)
49:     {
50:         return SDM_INVALID_MESSAGE;
51:     }
52:     cur += destination.Unmarshal(buf,cur);
53:     seconds = GET_ULONG (&buf[cur]);
54:     cur += sizeof(seconds);
55:     useconds = GET_ULONG (&buf[cur]);
56:     cur += sizeof(useconds);
57:     #ifdef BUILD_WITH_MESSAGE_LOGGING
58:     Logger.MessageReceived(*this);
59:     #endif
60:     return cur;
61: }
62:
63: long SDMTat::Send(const SDMComponent_ID& destination)
64: {
65:     int result;
66:     result = SendTo(destination);
67:     if (result <= 0)
68:         return SDM_MESSAGE_SEND_ERROR;
69:     return result;
70: }
71: int SDMTat::MsgToString(char *str_buf, int length) const
72: {
73:     char dest_id[40];
74:
75:     if (length < 8096)
76:         return 0;
77:     destination.IDToString(dest_id, 40);
78:     #ifdef WIN32
79:     sprintf(str_buf,
80:     #else

```

```
81: snprintf(str_buf,length,
82: #endif
83:      "SDMTat %ld:%ld %ld bytes, Dest: %s Seconds: %ld USeconds: %ld", sec, subsec, msg_length,
dest_id, seconds, useconds);
84: return strlen(str_buf);
85: }
```

File: sdm/common/message/SDMReady.h

```
1: #ifndef __SDM_READY_H_
2: #define __SDM_READY_H_
3:
4: #include "SDMmessage.h"
5:
6: //This class is intended for use by only SDM core components
7:
8: class SDMLIB_API SDMReady: public SDMmessage
9: {
10: public :
11:     SDMComponent_ID destination;          //The originator of the Ready message for sensor_id and
    port, and the address of node being sent to
12:     SDMComponent_ID source;              //The address of the sender, for heartbeat messages
13:     SDMReady();
14:     long Send();
15:     long Send(const SDMComponent_ID& destination);
16:     long Sendtcp(const SDMComponent_ID& destination);
17:     long SendBCast(long addr, unsigned short dest_port);
18:     long Marshal(char* buf);
19:     long Unmarshal(const char* buf);
20:     int MsgToString(char *str_buf, int lenght) const;
21: };
22:
23: #endif
```

File: sdm/common/message/SDMSerreqst.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMSerreqst.h"
4:
5:
SDMSerreqst::SDMSerreqst():source(),destination(),command_id(0,0),reply_id(0,0),fault_id(0,0),length(
0),seq_num(0)
6: {
7:   MsgName = SDM_Serreqst;
8:   data[0] = '\0';
9:   total_length = 28;
10: }
11:
12: long SDMSerreqst::Send(const SDMComponent_ID& destination)
13: {
14:   int result;
15:   result = SendTo(destination);
16:   if (result <= 0)
17:     return SDM_MESSAGE_SEND_ERROR;
18:   return result;
19: }
20:
21: long SDMSerreqst::Recv(long port)
22: {
23:   int result;
24:   result = RecvFrom(port);
25:   if (result <= 0)
26:     return SDM_MESSAGE_RECV_ERROR;
27:   return result;
28: }
29:
30: long SDMSerreqst::Marshal(char* buf)
31: {
32:   int cur;
33:   cur = HEADER_SIZE;
34:   cur += source.Marshal(buf,cur);
35:   cur += destination.Marshal(buf,cur);
36:   cur += command_id.Marshal(buf, cur);
37:   cur += reply_id.Marshal(buf, cur);
```

```

38: cur += fault_id.Marshal(buf, cur);
39: PUT_SHORT(&buf[cur],seq_num);
40: cur += sizeof(seq_num);
41: memcpy(buf+cur,&data,length);
42: cur += length;
43: msg_length = cur - HEADER_SIZE;
44: MarshalHeader(buf);
45: return cur;
46: }
47:
48: long SDMSerreqst::Unmarshal(const char* buf)
49: {
50: int cur;
51: cur = UnmarshalHeader(buf);
52: if(cur==SDM_INVALID_MESSAGE)
53: {
54:     return SDM_INVALID_MESSAGE;
55: }
56: if(total_length>msg_length)
57: {
58:     return SDM_INVALID_MESSAGE;
59: }
60: cur += source.Unmarshal(buf,cur);
61: cur += destination.Unmarshal(buf,cur);
62: cur += command_id.Unmarshal(buf, cur);
63: cur += reply_id.Unmarshal(buf, cur);
64: cur += fault_id.Unmarshal(buf, cur);
65: seq_num = GET_SHORT(&buf[cur]);
66: cur += sizeof(seq_num);
67: length = msg_length - cur + HEADER_SIZE;
68: memcpy(&data,buf+cur,length);
69: cur += length;
70: #ifdef BUILD_WITH_MESSAGE_LOGGING
71: Logger.MessageReceived(*this);
72: #endif
73: return cur;
74: }
75: int SDMSerreqst::MsgToString(char *str_buf, int length) const
76: {
77: char source_id[40];
78: char dest_id[40];

```



```

79: char data_section[511];
80: char cmd_id[30];
81: char rpy_id[30];
82: char flt_id[30];
83: int i, j;
84:
85: if (length < 8096)
86:     return 0;
87: for (i = 0, j = 0; j < this->length && i < 510; i+=2, j++)
88: {
89:     sprintf(&data_section[i], "%.2x", data[j]);
90: }
91: data_section[i] = '\0';
92:
93: source.IDToString(source_id, 40);
94: destination.IDToString(dest_id, 40);
95: command_id.IDToString(cmd_id, 30);
96: reply_id.IDToString(rpy_id, 30);
97: fault_id.IDToString(flt_id, 30);
98: #ifdef WIN32
99: sprintf(str_buf,
100: #else
101:     snprintf(str_buf, length,
102: #endif
103:     "SDMSerreqst %ld:%ld %ld bytes, Source: %s Dest: %s CommandID: %s ReplyID: %s
FaultID: %s seq_num: %d [DATA SECTION: %s ]", sec, subsec, msg_length, source_id, dest_id,
cmd_id, rpy_id, flt_id, seq_num, data_section);
104:     return strlen(str_buf);
105: }

```

File: sdm/common/message/SDMReqxTEDS.h

```
1: #ifndef __SDM_REQXTEDS_H_
2: #define __SDM_REQXTEDS_H_
3:
4: #include "SDMmessage.h"
5:
6: class SDMLIB_API SDMReqxTEDS: public SDMmessage
7: {
8: public :
9:     char select;           //indicates whether sensor id or device name is used
10:    SDMComponent_ID destination; //the subscriber who wants the xTEDS
11:    SDMComponent_ID source;    //unique id of xTEDS provider
12:    char device_name[XTEDS_MAX_ITEM_NAME_SIZE]; //device name of xTEDS provider
13:    SDMReqxTEDS();
14:    long Send();
15:    long Recv(long port); SDM_DEPRECATED
16:    long Marshal(char* buf);
17:    long Unmarshal(const char* buf);
18:    int MsgToString(char *str_buf, int length) const;
19: };
20:
21: #endif
```

File: sdm/common/message/SDMRegister.h

```
1: #ifndef __SDM_REGISTER_H_
2: #define __SDM_REGISTER_H_
3:
4: #include "SDMmessage.h"
5:
6:
7: class SDMLIB_API SDMRegister : public SDMmessage
8: {
9: public:
10:     short sensorIndex;
11:
12:     SDMRegister();
13:     long Marshal(char *buf);
14:     long Unmarshal(const char *buf);
15:     int MsgToString(char *str_buf, int length) const;
16: private:
17: };
18:
19: #endif
```

File: sdm/common/message/SDMmessage.h

```
1: #ifndef __SDM_MESSAGE_H_
2: #define __SDM_MESSAGE_H_
3:
4:
5: // #include <byteswap.h> // Macros for moving bytes around if necessary
6: #include "../message_defs.h"
7: #include "../marshall.h"
8: #include "SDMComponent_ID.h"
9: #include "../sdmLib.h"
10: #include "../UDPcom.h"
11: #ifdef BUILD_WITH_MESSAGE_LOGGING
12: # include "../MessageLogger/SDMMessageLogger.h"
13: #endif
14:
15: #define SDM_OK      0
16: #define SDM_MESSAGE_SEND_ERROR -1
17: #define SDM_MESSAGE_RECV_ERROR -2
18: #define SDM_INVALID_MESSAGE -3
19: #define SDM_NO_FURTHER_DATA_PROVIDER -4
20: #define SDM_COULD_NOT_FORK -5
21: #define SDM_NO_FURTHER_XTEDS -6
22: #define SDM_ASYNC_PIPE_ERROR -7
23: #define SDM_INVALID_SELECT -8
24: #define SDM_UNABLE_TO_REGISTER -9
25: #define SDM_INVALID_XTEDS -10
26: #define SDM_UNKNOWN_XTEDS -11
27: #define SDM_INVALID_UPDATE -12
28: #define SDM_UNABLE_TO_UPDATE -13
29: #define SDM_INVALID_CANCEL -14
30: #define SDM_PM_NOT_AVAILABLE -15
31: // Header size for all SDM* messages
32: #define HEADER_SIZE 11
33:
34: enum SDM_TYPE
35: {
36:     _INTEGER,
37:     _LONG,
38:     _CHAR,
39:     _BYTE,
```

```

40: //_STRING,
41: _FLOAT
42: };
43:
44: extern SDMLIB_API SDMComponent_ID TaskManager;
45: extern SDMLIB_API SDMComponent_ID DataManager;
46:
47: extern SDMLIB_API int PID;
48: extern SDMLIB_API long getPort();
49: extern SDMLIB_API void SDMInit(int argc,char** argv);
50:
51: class SDMLIB_API SDMmessage
52: {
53: protected:
54: char MsgName;           //byte indicating the type of message
55: long sec;               //seconds portion of timestamp
56: long subsec;           //subseconds portion
57: short msg_length;      //msg_length of message not including header (consistant with SPA-U)
58: short total_length;    //a known or minimum length for a given message not including the
header
59: public:
60: SDMmessage():MsgName('
\0'),sec(0),subsec(0),msg_length(0),total_length(0),bound(IP_SOCKET_INVALID){}
61: virtual ~SDMmessage() {}
62: virtual long Marshal(char* buf) =0;
63: virtual long Unmarshal(const char* buf)=0;
64: virtual int MsgToString(char *str_buf, int length) const =0;
65: bool Avail(long port, int timeout) const;
66: bool Avail(long port) const;
67: long SendTo(const SDMComponent_ID& destination);
68: char GetMsgName() const;
69: long Forward(const SDMComponent_ID& destination);
70: long GetSecondsStamp () const { return sec; }
71: long GetSubSecondsStamp () const { return subsec; }
72: protected:
73: long MarshalHeader(char* buf);
74: long UnmarshalHeader(const char* buf);
75: long SendDM();
76: long SendTM();
77: long RecvFrom(long port); /*__attribute__ ((deprecated));*/
78: long SendTCP(long ip_addr,long port);

```

```

79: long SendBroadcast(long address=0, unsigned short dest_port=PORT_DM_ELECTION);
80: long SendReplyTo(int socket, const struct sockaddr_in* sin, bool tcp);
81: // bound is deprecated to be removed with RecvFrom()
82: int bound;          //the socket the class currently has access to or -1 if none
83: #ifdef BUILD_WITH_MESSAGE_LOGGING
84: static SDMMessagesLogger Logger;
85: #endif
86: private:
87: long MarshalHeaderOldTimeStamp(char* buf);
88: };
89:
90: #endif // __SDM_MESSAGE_H_

```

File: sdm/common/message/SDMComponent_ID.h

```
1: #ifndef __SDM_COMPONENT_ID_H_
2: #define __SDM_COMPONENT_ID_H_
3: #include <arpa/inet.h>
4:
5: #include "../sdmLib.h"
6:
7: const unsigned long ADDR_LOCAL_HOST = inet_addr("127.0.0.1");
8:
9: class SDMLIB_API SDMComponent_ID
10: {
11: public:
12: SDMComponent_ID();
13: SDMComponent_ID(const SDMComponent_ID&);
14: ~SDMComponent_ID();
15:
16: SDMComponent_ID& operator=(const SDMComponent_ID&);
17: bool operator==(const SDMComponent_ID&) const;
18:
19: int Marshal(char* buf, int start) const;
20: int Unmarshal(const char* buf, int start);
21:
22: void setSensorID(unsigned long new_sensor_id) { m_ulSensorId = new_sensor_id; }
23: void setAddress(unsigned long new_address) { m_ulAddress = new_address; }
24: void setPort(unsigned short new_port) { m_usPort = new_port; }
25: int IDToString(char *buf, int length) const;
26:
27: unsigned long getSensorID() const { return m_ulSensorId; }
28: unsigned long getAddress() const { return m_ulAddress; }
29: unsigned short getPort() const { return m_usPort; }
30:
31: bool isEmpty() const;
32:
33: private:
34: unsigned long m_ulSensorId;    //The sensor_id assigned by the Data Manager or your local id
35: unsigned long m_ulAddress;     //The ip address of the node the device or application is running
    on
36: unsigned short m_usPort;      //The port that is being listened on for incoming messages
37: };
38:
```

```
39: static const SDMComponent_ID COMPONENT_ID_INVALID;  
40:  
41: #endif
```


File: sdm/common/message/SDMReqxTEDS.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMReqxTEDS.h"
4:
5: SDMReqxTEDS::SDMReqxTEDS():select(-1),destination(),source()
6: {
7:     MsgName = SDM_ReqxTEDS;
8:     device_name[0] = '\0';
9:     total_length = 12;
10: }
11:
12: long SDMReqxTEDS::Send()
13: {
14:     return SendDM();
15: }
16:
17: long SDMReqxTEDS::Recv(long port)
18: {
19:     int result;
20:     result = RecvFrom(port);
21:     if (result <= 0)
22:         return SDM_MESSAGE_RECV_ERROR;
23:     return result;
24: }
25:
26: long SDMReqxTEDS::Marshal(char* buf)
27: {
28:     int cur;
29:     cur = HEADER_SIZE;
30:     cur += destination.Marshal(buf,cur);
31:     PUT_CHAR(&buf[cur],select);
32:     cur += sizeof(select);
33:     switch(select)
34:     {
35:     case 0:
36:     case 2: //use sensor_id
37:         cur += source.Marshal(buf,cur);
38:         break;
39:     case 1: //use device_name
```

```

40: case 3:
41:     msg_length = (short)strlen(device_name);
42:     memcpy(buf+cur,device_name,msg_length);
43:     cur += msg_length;
44:     buf[cur] = '\0';
45:     cur++;
46:     break;
47: default:
48:     return SDM_INVALID_SELECT;
49: }
50: msg_length = cur - HEADER_SIZE;
51: MarshalHeader(buf);           //re-marshal header with correct length
52: return msg_length + HEADER_SIZE;
53: }
54:
55: long SDMReqxTEDS::Unmarshal(const char* buf)
56: {
57: int cur;
58: long device_name_msg_length;
59:
60: cur = UnmarshalHeader(buf);
61: if(cur==SDM_INVALID_MESSAGE)
62: {
63:     return SDM_INVALID_MESSAGE;
64: }
65: if(total_length>msg_length)
66: {
67:     return SDM_INVALID_MESSAGE;
68: }
69: cur += destination.Unmarshal(buf,cur);
70: select = GET_CHAR (&buf[cur]);
71: cur += sizeof(select);
72: switch(select)
73: {
74: case 0:
75: case 2:
76:     cur += source.Unmarshal(buf,cur);
77:     return cur;
78: case 1:
79: case 3:
80:     device_name_msg_length = (long)strlen(buf+cur);

```

```

81:     memset(device_name,0,sizeof(device_name));
82:     memcpy(&device_name,buf+cur,device_name_msg_length);
83:     cur += device_name_msg_length + 1;
84:     return cur;
85: default:
86:     break;
87: };
88: #ifdef BUILD_WITH_MESSAGE_LOGGING
89: Logger.MessageReceived(*this);
90: #endif
91: return SDM_INVALID_SELECT;
92: }
93: int SDMReqxTEDS::MsgToString(char *str_buf, int length) const
94: {
95:     char source_id[40];
96:     char dest_id[40];
97:
98:     if (length < 8096)
99:         return 0;
100:     source.IDToString(source_id, 40);
101:     destination.IDToString(dest_id, 40);
102: #ifdef WIN32
103:     sprintf(str_buf,
104: #else
105:     snprintf(str_buf,length,
106: #endif
107:     "SDMReqxTEDS %ld:%ld %ld bytes, Select: %d", sec, subsec, msg_length, select);
108:     switch(select)
109:     {
110:     case 0:
111:     case 2: //use sensor_id
112:         strcat(str_buf," Source: ");
113:         strcat(str_buf, source_id);
114:         break;
115:     case 1: //use device_name
116:     case 3:
117:         strcat(str_buf," DeviceName: ");
118:         strcat(str_buf, device_name);
119:         break;
120:     default:
121:         break;

```

```
122:  }  
123:  return (int)strlen(str_buf);  
124: }
```

File: sdm/common/message/SDMComponent_ID.cpp

```
1: #include "SDMComponent_ID.h"
2: #include "../marshall.h"
3:
4: #include <stdlib.h>
5: #include <string.h>
6: #include <arpa/inet.h>
7: #include <sys/socket.h>
8: #include <netinet/in.h>
9: #include <stdio.h>
10:
11: SDMComponent_ID::SDMComponent_ID() : m_ulSensorId(0), m_ulAddress(0), m_usPort(0)
12: {
13: }
14:
15: SDMComponent_ID::SDMComponent_ID(const SDMComponent_ID& b) :
m_ulSensorId(b.m_ulSensorId), m_ulAddress(b.m_ulAddress), m_usPort(b.m_usPort)
16: {
17: }
18:
19: SDMComponent_ID::~SDMComponent_ID()
20: {
21: }
22:
23: SDMComponent_ID& SDMComponent_ID::operator=(const SDMComponent_ID& b)
24: {
25: m_ulSensorId = b.m_ulSensorId;
26: m_ulAddress = b.m_ulAddress;
27: m_usPort = b.m_usPort;
28: return *this;
29: }
30:
31: bool SDMComponent_ID::operator==(const SDMComponent_ID& b) const
32: {
33: if(m_ulSensorId == b.m_ulSensorId)
34: {
35: if(m_ulAddress == b.m_ulAddress)
36: {
37: if(m_usPort == b.m_usPort)
38: {
```

```

39:         return true;
40:     }
41: }
42: }
43: return false;
44: }
45:
46: int SDMComponent_ID::Marshal(char* buf, int start) const
47: {
48:     int cur = 0;
49:
50:     // Marshal the address. On either little or big endian archs, m_ulAddress is always in
51:     // network byte order (big endian), so perform a swap and copy the result into the message
52:     // buffer using SDM message byte order (little endian).
53:     unsigned long ulSwappedAddr = bswap_32(m_ulAddress); // Ip address in little-endian
54:     *reinterpret_cast<unsigned long*>(&buf[start]) = ulSwappedAddr;
55:
56:     cur += sizeof(ulSwappedAddr);
57:     PUT_USHORT(&buf[start+cur], m_usPort);
58:     cur += sizeof(m_usPort);
59:     PUT_ULONG(&buf[start+cur], m_ulSensorId);
60:     cur += sizeof(m_ulSensorId);
61:     return cur;
62: }
63:
64: int SDMComponent_ID::Unmarshal(const char* buf, int start)
65: {
66:     int cur = 0;
67:
68:     // Unmarshal the address. On either little or big endian archs, the address in the buffer is
69:     // in SDM message byte order (little endian), m_ulAddress is to be maintained in network byte
70:     // order (big endian), so it must be swapped identically for each target.
71:     unsigned long ulSwappedAddr = static_cast<unsigned long>(*reinterpret_cast<const unsigned long*>(&buf[start]));
72:     m_ulAddress = bswap_32(ulSwappedAddr);
73:
74:     cur += sizeof(m_ulAddress);
75:     m_usPort = GET_USHORT(&buf[start+cur]);
76:     cur += sizeof(m_usPort);
77:     m_ulSensorId = GET_ULONG(&buf[start+cur]);
78:     cur += sizeof(m_ulSensorId);

```

```

79: return cur;
80: }
81:
82: /*
83: NOW INLINE
84: void SDMComponent_ID::setSensorID(unsigned long new_sensor_id)
85: {
86:     m_ulSensorId = new_sensor_id;
87: }
88:
89: void SDMComponent_ID::setAddress(unsigned long new_address)
90: {
91:     m_ulAddress = new_address;
92:     BigEndian = true;
93: }
94:
95: void SDMComponent_ID::setPort(unsigned short new_port)
96: {
97:     m_usPort = new_port;
98: }
99:
100: unsigned long SDMComponent_ID::getSensorID() const
101: {
102:     return m_ulSensorId;
103: }
104:
105: unsigned long SDMComponent_ID::getAddress() const
106: {
107:     return m_ulAddress;
108: }
109:
110: unsigned short SDMComponent_ID::getPort() const
111: {
112:     return m_usPort;
113: }*/
114:
115: bool SDMComponent_ID::isEmpty() const
116: {
117:     bool result = false;
118:
119:     if(m_ulSensorId == 0)

```

```

120:  {
121:      if(m_ulAddress == 0)
122:      {
123:          if(m_usPort == 0)
124:          {
125:              result = true;
126:          }
127:      }
128:  }
129:  return result;
130: }
131:
132: /*
133:  * IDToString() puts the component id into human readable form for debug purposes.
134:  * The length parameter and size of the buffer should be at least 36
135:  */
136: int SDMComponent_ID::IDToString(char *buf, int length) const
137: {
138:     if (length < 36)
139:         return 0;
140:     struct in_addr address;
141:     address.s_addr = m_ulAddress;
142: #ifdef WIN32
143:     sprintf(buf, "%s:%d SID=%ld",inet_ntoa(address), m_usPort, m_ulSensorId);
144: #else
145:     snprintf(buf, length,"%s:%d SID=%ld",inet_ntoa(address), m_usPort, m_ulSensorId);
146: #endif
147:     return (int)strlen(buf);
148: }

```


File: sdm/common/message/SDMDeletesub.cpp

```
1: #include "SDMDeletesub.h"
2: #include <stdio.h>
3: #include <string.h>
4:
5: SDMDeletesub::SDMDeletesub():source(),destination(),msg_id(0,0)
6: {
7:     MsgName = SDM_Deletesub;
8:     total_length = 22;
9: }
10:
11: long SDMDeletesub::Send()
12: {
13:     int result;
14:     result = SendTo(source);
15:     if (result <= 0)
16:         return SDM_MESSAGE_SEND_ERROR;
17:     return result;
18: }
19:
20: long SDMDeletesub::Recv(long port)
21: {
22:     int result;
23:     result = RecvFrom(port);
24:     if (result <= 0)
25:         return SDM_MESSAGE_RECV_ERROR;
26:     return result;
27: }
28:
29: long SDMDeletesub::Marshal(char* buf)
30: {
31:     int cur = HEADER_SIZE;
32:     cur += source.Marshal(buf,cur);
33:     cur += destination.Marshal(buf,cur);
34:     cur += msg_id.Marshal(buf, cur);
35:     msg_length = cur - HEADER_SIZE;
36:     MarshalHeader(buf);
37:     return cur;
38: }
39:
```

```

40: long SDMDeletesub::Unmarshal(const char* buf)
41: {
42: int cur;
43: cur = UnmarshalHeader(buf);
44: if(cur == SDM_INVALID_MESSAGE)
45: {
46:     return SDM_INVALID_MESSAGE;
47: }
48: if(total_length!=msg_length)
49: {
50:     return SDM_INVALID_MESSAGE;
51: }
52: cur += source.Unmarshal(buf,cur);
53: cur += destination.Unmarshal(buf,cur);
54: cur += msg_id.Unmarshal(buf, cur);
55: #ifdef BUILD_WITH_MESSAGE_LOGGING
56: Logger.MessageReceived(*this);
57: #endif
58: return HEADER_SIZE+msg_length;
59: }
60: int SDMDeletesub::MsgToString(char *str_buf, int length) const
61: {
62: char source_id[40];
63: char dest_id[40];
64: char message_id[30];
65:
66: if (length < 8096)
67:     return 0;
68: source.IDToString(source_id, 40);
69: destination.IDToString(dest_id, 40);
70: msg_id.IDToString(message_id, 30);
71: #ifdef WIN32
72: sprintf(str_buf,"SDMDeletesub %ld:%ld %d bytes, Source: %s Dest: %s %s", sec, subsec,
msg_length, source_id, dest_id, message_id);
73: #else
74: snprintf(str_buf,length,"SDMDeletesub %ld:%ld %d bytes, Source: %s Dest: %s %s", sec, subsec,
msg_length, source_id, dest_id, message_id);
75: #endif
76: return (int)strlen(str_buf);
77: }

```

File: sdm/common/message/SDMxTEDS.h

```
1: #ifndef __SDM_XTEDS_H_
2: #define __SDM_XTEDS_H_
3:
4: #include "SDMmessage.h"
5:
6: #define FILENAME_SIZE 85           // maximum filename length
7:
8: #define MAX_XTEDS_SEQUENCE_VALUE 15
9: #define MAX_USB_PATH_SIZE 80
10: #define MAX_XTEDS_SIZE (LARGE_MSG_BUFSIZE - 105)
11:
12:
13: class SDMLIB_API SDMxTEDS: public SDMmessage
14: {
15: public :
16:     SDMComponent_ID source;        //unique sensor id, applications need only set least significant
byte
17:     char active;                   //status of the xTEDS (active or inactive), default is active
18:     char xTEDS[MAX_XTEDS_SIZE];    //the xTEDS to be registered with the SDM system
19:     char SPA_node[MAX_USB_PATH_SIZE]; //path through USB network to device from sensor
manager, NULL for applications
20:     long pid;                      //SDM process id of xTEDS provider
21:     SDMxTEDS();
22:     long Send();
23:     long Recv(long port); SDM_DEPRECATED
24:     long Marshal(char* buf);
25:     long Unmarshal(const char* buf);
26:     long getPID(void);
27:     void setPID(void);
28:     int MsgToString(char *str_buf, int length) const;
29: };
30:
31: #endif
```

File: sdm/common/message/SDMKill.h

```
1: #ifndef __SDM_KILL_H_
2: #define __SDM_KILL_H_
3:
4: #include "SDMmessage.h"
5:
6: class SDMLIB_API SDMKill: public SDMmessage
7: {
8: public :
9:     unsigned long PID;      //The pid of the process to kill
10:    unsigned char killLevel; //0 to restart process if data is requested, 1 to completely kill task
11:    SDMKill();
12:    long Send();
13:    long Send(const SDMComponent_ID& destination);
14:    long Marshal(char* buf);
15:    long Unmarshal(const char* buf);
16:    int MsgToString(char *str_buf, int lenght) const;
17: };
18:
19: #endif
```

File: sdm/common/message/SDMConsume.h

```
1: #ifndef __SDM_CONSUME_H_
2: #define __SDM_CONSUME_H_
3:
4: #include "SDMmessage.h"
5: #include "SDMMessage_ID.h"
6:
7: class SDMLIB_API SDMConsume: public SDMmessage
8: {
9: public :
10:
11:     SDMComponent_ID source;           //the data provider
12:     SDMComponent_ID destination;      //the subscriber
13:     SDMMessage_ID msg_id;             //unique id of desired data message as defined in the xTEDS of
    provider
14:     SDMConsume();
15:     long Send();
16:     long Marshal(char* buf);
17:     long Unmarshal(const char* buf);
18:     int MsgToString(char *str_buf, int length) const;
19: };
20:
21: #endif
```

File: sdm/common/message/SDMHello.h

```
1: #ifndef __SDM_HELLO_H_
2: #define __SDM_HELLO_H_
3: #include "SDMmessage.h"
4:
5:
6:
7: class SDMLIB_API SDMHello : public SDMmessage
8: {
9: public:
10: SDMComponent_ID source;
11: char type;          //Component Type. 'D'=Device, 'A'=Application
12:
13: SDMHello();
14: long Send();
15: long Marshal(char *buf);
16: long Unmarshal(const char *buf);
17: int MsgToString(char *buf, int length) const;
18: };
19:
20: #endif
```

File: sdm/common/message/SDMxTEDSInfo.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMxTEDSInfo.h"
4: #include "../UDPcom.h"
5:
6: SDMxTEDSInfo::SDMxTEDSInfo():source(),emptyflag(false)
7: {
8:     MsgName = SDM_xTEDSInfo;
9:     xTEDS[0] = '\0';
10:    total_length = HEADER_SIZE;
11: }
12:
13: long SDMxTEDSInfo::Send()
14: {
15:    return SendDM();
16: }
17:
18: long SDMxTEDSInfo::Marshal(char* buf)
19: {
20:    int cur;
21:    if(emptyflag == true)
22:    {
23:        cur = MarshalEmpty(buf);
24:        return cur;
25:    }
26:    cur = HEADER_SIZE;
27:    long i= strlen(xTEDS);
28:    cur += source.Marshal(buf,cur);
29:    memcpy(buf+cur,xTEDS,i+1);
30:    cur += i + 1;
31:    msg_length = cur - HEADER_SIZE;
32:    MarshalHeader(buf);
33:    return cur;
34: }
35:
36: long SDMxTEDSInfo::Unmarshal(const char* buf)
37: {
38:    int cur;
39:    cur = UnmarshalHeader(buf);
```

```

40: if(cur==SDM_INVALID_MESSAGE)
41: {
42:     return SDM_INVALID_MESSAGE;
43: }
44: if(msg_length == 0)
45:     return SDM_NO_FURTHER_XTEDS;
46: if(total_length>msg_length)
47: {
48:     return SDM_INVALID_MESSAGE;
49: }
50: cur += source.Unmarshal(buf,cur);
51: //
52: // Get the xTEDS
53: unsigned int uiCurLength = strlen(buf + cur);
54: strncpy(xTEDS, buf + cur, sizeof(xTEDS));
55: if (uiCurLength > sizeof(xTEDS) - 1)
56:     xTEDS[sizeof(xTEDS) - 1] = '\0';
57:
58: #ifdef BUILD_WITH_MESSAGE_LOGGING
59: Logger.MessageReceived(*this);
60: #endif
61: return msg_length + HEADER_SIZE;
62: }
63:
64: long SDMxTEDSInfo::Send(const SDMComponent_ID& destination)
65: {
66: int result;
67: emptyflag = false;
68: result = SendTo(destination);
69: if (result <= 0)
70:     return SDM_MESSAGE_SEND_ERROR;
71: return result;
72: }
73:
74: long SDMxTEDSInfo::Recv(long port)
75: {
76: int result;
77: result = RecvFrom(port);
78: if (result <= 0)
79:     return SDM_MESSAGE_RECV_ERROR;
80: return result;

```



```

81: }
82:
83: long SDMxTEDSInfo::MarshalEmpty(char* buf)
84: {
85: int cur;
86: msg_length = 0;
87: cur = MarshalHeader(buf);
88: return cur;
89: }
90:
91: long SDMxTEDSInfo::SendEmpty(const SDMComponent_ID& destination)
92: {
93: int result;
94: emptyflag = true;
95: result = SendTo(destination);
96: if (result <= 0)
97:     return SDM_MESSAGE_SEND_ERROR;
98: return result;
99: }
100:
101: int SDMxTEDSInfo::MsgToString(char *str_buf, int length) const
102: {
103:     char source_id[40];
104:
105:     if (length < 8096)
106:         return 0;
107:     source.IDToString(source_id, 40);
108: #ifdef WIN32
109:     sprintf(str_buf,
110: #else
111:     snprintf(str_buf,length,
112: #endif
113:         "SDMxTEDSInfo %ld:%ld %ld bytes, Source: %s xTEDS: %s", sec, subsec, msg_length,
source_id, xTEDS);
114:     return strlen(str_buf);
115: }

```

File: sdm/common/message/SDMKill.cpp

```
1: #include <string.h>
2: #include <stdio.h>
3: #include "SDMKill.h"
4: #include "../UDPcom.h"
5:
6: SDMKill::SDMKill():PID(0), killLevel(0)
7: {
8:     MsgName = SDM_Kill;
9:     total_length = 5;
10: }
11:
12: long SDMKill::Send(const SDMComponent_ID& destination)
13: {
14:     return SendTo(destination);
15: }
16:
17: long SDMKill::Send()
18: {
19:     return SendTo(TaskManager);
20: }
21:
22: long SDMKill::Marshal(char* buf)
23: {
24:     int cur = HEADER_SIZE;
25:     PUT_ULONG(&buf[cur],PID);
26:     cur += sizeof(PID);
27:     PUT_UCHAR(&buf[cur], killLevel);
28:     cur += sizeof(killLevel);
29:     msg_length = cur - HEADER_SIZE;
30:     MarshalHeader(buf);
31:     return cur;
32: }
33:
34: long SDMKill::Unmarshal(const char* buf)
35: {
36:     int cur;
37:     cur = UnmarshalHeader(buf);
38:     if(cur == SDM_INVALID_MESSAGE)
39: {
```

```

40:     return SDM_INVALID_MESSAGE;
41: }
42: if(msg_length == 4)
43: {
44:     PID = GET_ULONG(&buf[cur]);
45:     cur += sizeof(PID);
46:     killLevel = 0;
47: }
48: else
49: {
50:     if(total_length!=msg_length)
51:     {
52:         return SDM_INVALID_MESSAGE;
53:     }
54:     PID = GET_ULONG(&buf[cur]);
55:     cur += sizeof(PID);
56:     killLevel = GET_UCHAR(&buf[cur]);
57:     cur += sizeof(killLevel);
58: }
59: #ifdef BUILD_WITH_MESSAGE_LOGGING
60: Logger.MessageReceived(*this);
61: #endif
62: return HEADER_SIZE+msg_length;
63: }
64: int SDMKill::MsgToString(char *str_buf, int length) const
65: {
66:     if (length < 8096)
67:         return 0;
68: #ifdef WIN32
69:     sprintf(str_buf,
70: #else
71:     snprintf(str_buf,length,
72: #endif
73:         "SDMKill %ld:%ld %ld bytes, PID: %ld",sec,subsec,msg_length,PID);
74:     return (int)strlen(str_buf);
75: }

```

File: sdm/common/xTEDS/xTEDSOrientationList.cpp

```
1: #include <string.h>
2: #include "xTEDSOrientationList.h"
3: #include "MessageDef.h"
4:
5: xTEDSOrientationList::xTEDSOrientationList():Head(NULL)
6: {
7: }
8:
9: xTEDSOrientationList::~~xTEDSOrientationList()
10: {
11: DeleteList();
12: }
13:
14: void xTEDSOrientationList::AddOrientation(const orientation* NewOrientation)
15: {
16: xTEDSOrientationItemNode *Cur;
17:
18: if (NewOrientation == NULL) return;
19:
20: if (Head == NULL)
21: {
22:     Head = new xTEDSOrientationItemNode();
23:     Head->OrientationData.SetOrientation(NewOrientation);
24: }
25: else
26: {
27:     for (Cur = Head; Cur->Next != NULL; Cur = Cur->Next)
28:         ;
29:     xTEDSOrientationItemNode *NewItem = new xTEDSOrientationItemNode();
30:     NewItem->OrientationData.SetOrientation(NewOrientation);
31:     Cur->Next = NewItem;
32: }
33: }
34:
35: void xTEDSOrientationList::SetOrientation(const orientation* OrientationList)
36: {
37: // Add the head first
38: AddOrientation(OrientationList);
39: // Add the rest
```

```

40: for (orientation* Cur = OrientationList->next; Cur != NULL; Cur = Cur->next)
41: {
42:     AddOrientation(Cur);
43: }
44: }
45:
46: void xTEDSOrientationList::DeleteList()
47: {
48: for (xTEDSOrientationItemNode *Cur = Head; Cur != NULL; )
49: {
50:     xTEDSOrientationItemNode *Temp = Cur->Next;
51:     delete Cur;
52:     Cur = Temp;
53: }
54: Head = NULL;
55: }
56:
57: void xTEDSOrientationList::VarInfoRequest(char* InfoBufferOut, size_t BufferLength) const
58: {
59: const int MAX_BUF_SIZE = 1024;
60: char Buf[MAX_BUF_SIZE];
61: size_t BufLength = 0;
62:
63: Buf[0] = '\0';
64: for (xTEDSOrientationItemNode *Cur = Head; Cur != NULL; Cur = Cur->Next)
65: {
66:     BufLength = strlen(Buf);
67:     strncat(Buf, "\n\t", sizeof(Buf) - BufLength);
68:
69:     BufLength += 2;
70:     Cur->OrientationData.VarInfoRequest(Buf + BufLength, sizeof(Buf) - BufLength);
71: }
72: strncat(InfoBufferOut, Buf, BufferLength - 1);
73: }

```

File: sdm/common/xTEDS/xTEDSMessage.cpp

```
1: #include "xTEDSMessage.h"
2: #include "xTEDSItem.h"
3: #include "xTEDSVariableList.h"
4:
5: #include <stdlib.h>
6:
7: xTEDSMessage::xTEDSMessage():m_Variables()
8: {
9: }
10:
11: xTEDSMessage::~~xTEDSMessage()
12: {
13: }
14:
15: bool xTEDSMessage::addVariable(xTEDSVariable* pVar)
16: {
17: if (pVar == NULL) return false;
18: m_Variables.addItem(pVar);
19: return true;
20: }
21:
22: int xTEDSMessage::getMessageID() const
23: {
24: return m_ItemInterfaceMessageID.getMessage();
25: }
26:
27: void xTEDSMessage::setMessageID(int NewID)
28: {
29: if (NewID > 0 && NewID < 256)
30:     m_ItemInterfaceMessageID.setMessage(static_cast<unsigned char>(NewID));
31: }
32:
33: VariableDef* xTEDSMessage::GetVariableXtedsDefinitions() const
34: {
35:     SDMMMessage_ID defaultMessageId(0,0);
36:
37:     // Request VarInfo for all of the variables
38:     return m_Variables.VarInfoRequest("", defaultMessageId, xTEDSVariableList::MATCH_ALL);
39: }
```

```
40:
41: #ifndef REMOVE_DEBUG_OUTPUT
42: void xTEDSMessage::PrintDebug() const
43: {
44:     printf(" ");
45:     xTEDSItem::PrintDebug();
46: }
47: #endif
48:
```

File: sdm/common/xTEDS/xTEDSCurve.h

```
1: #ifndef __SDM_XTEDS_CURVE_H_
2: #define __SDM_XTEDS_CURVE_H_
3:
4: #include "../message_defs.h"
5: extern "C"
6: {
7: #include "xTEDSParser.h"
8: }
9: #include "xTEDSCoefList.h"
10:
11: class xTEDSCurve
12: {
13: public:
14: xTEDSCurve();
15: xTEDSCurve(const xTEDSCurve&);
16:
17: ~xTEDSCurve();
18:
19: void setCurve(const curve* NewCurve);
20: void setName(const char*);
21: void setDescription(const char*);
22:
23: void VarInfoRequest(char* InfoBufferOut, size_t BufferSize) const;
24: private:
25: xTEDSCurve& operator=(const xTEDSCurve&);
26: char* m_strName;
27: char* m_strDescription;
28: xTEDSCoefList m_xclCoefs;
29: };
30:
31: #endif
```


File: sdm/common/xTEDS/xTEDSVariableList.h

```
1: #ifndef __SDM_XTEDS_VARIABLE_LIST_H_
2: #define __SDM_XTEDS_VARIABLE_LIST_H_
3:
4: //The xTEDSVariableList does not 'own' its items, it is just a way for items to reference other items in
the tree
5: //The xTEDSItemTree however does 'own' its items, and deletes its xTEDSItems in its destructor
6:
7: #include "xTEDSVariable.h"
8: #include "xTEDSQualifierList.h"
9: #include "VariableDef.h"
10: #include "../message/SDMMessage_ID.h"
11:
12: class xTEDSVariableListNode
13: {
14: public:
15: xTEDSVariableListNode(const xTEDSVariable *NewData) : data(NewData), next(NULL) {}
16: const xTEDSVariable* data;
17: class xTEDSVariableListNode* next;
18: };
19:
20: class xTEDSVariableList
21: {
22: public:
23:     enum VarInfoMatchType { MATCH_VAR_NAME, MATCH_ALL };
24: xTEDSVariableList();
25: xTEDSVariableList(const xTEDSVariableList&);
26: ~xTEDSVariableList();
27: void addItem(const xTEDSVariable*);
28: void DeleteItems();
29: xTEDSVariableList& operator=(const xTEDSVariableList&);
30: void VarReqReg(char* InfoBufferOut, int BufferSize) const;
31: void VarRef(char* InfoBufferOut, int BufferSize) const;
32: VariableDef* VarInfoRequest(const char* VarName, const SDMMessage_ID& Interface,
33:     VarInfoMatchType matchType = MATCH_VAR_NAME) const;
34: MessageDef* RegInfo() const;
35: bool RegInfoMatch(const char* name, const xTEDSQualifierList&, const char* interface) const;
36: bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char* Interface)
const;
37: const xTEDSVariable* Find(const char* name, const char* interface = NULL) const;
```

```
38: bool IsEmpty() const;
39:
40: void PrintDebug() const;
41: private:
42: xTEDSVariableListNode* head;
43: xTEDSVariableListNode* tail;
44: };
45:
46: #endif
```

File: sdm/common/xTEDS/xTEDSParser.h

```
1: #ifndef __XTEDS_PARSER_H
2: #define __XTEDS_PARSER_H
3: /*contains support routines for the xTEDS parser*/
4:
5: #include "../sdmLib.h"
6:
7: typedef struct qualifier_data
8: {
9:   char* name;
10:  char* value;
11:  char* units;
12:  struct qualifier_data* next;
13: }qualifier_type;
14:
15: typedef struct coefficient_data
16: {
17:  char* exponent;
18:  char* value;
19:  char* description;
20:  struct coefficient_data* next;
21: }coef;
22:
23: typedef struct curve_data
24: {
25:  char* name;
26:  char* description;
27:  coef* coefs;
28: }curve;
29:
30: typedef struct option_data
31: {
32:  char* name;
33:  char* value;
34:  char* description;
35:  char* alarm;
36:  struct option_data* next;
37: }curveoption;
38:
39: typedef struct drange_data
```

```

40: {
41: char* name;
42: char* description;
43: curveoption* options;
44: }drange;
45:
46: typedef struct location_data
47: {
48: char *x_value;
49: char *y_value;
50: char *z_value;
51: char *units;
52: }location;
53:
54: typedef struct orientation_data
55: {
56: char *axis_value;
57: char *angle_value;
58: char *units_value;
59: struct orientation_data* next;
60: }orientation;
61:
62: typedef struct variable_data
63: {
64: char* length;
65: char* kind;
66: char* name;
67: char* qualifier;
68: char* id;
69: char* range_min;
70: char* range_max;
71: char* default_value;
72: char* precision;
73: char* units;
74: char* accuracy;
75: char* scale_factor;
76: char* scale_units;
77: char* format;
78: char* description;
79: char* interface_name;
80: char* interface_id;

```

```

81: char* r_low;
82: char* r_high;
83: char* y_low;
84: char* y_high;
85: char* invalid_value;
86: qualifier_type* qualifiers;
87: curve* curves;
88: drange* dranges;
89: struct variable_data* next;
90: location* location_data;
91: orientation* orientation_data;
92: } variable;
93:
94: typedef struct variable_reference
95: {
96: char* name;
97: struct variable_reference* next;
98: } var_ref;
99:
100: typedef struct fault_message
101: {
102:     char* name;
103:     char* id;
104:     char* description;
105:     char* interface_name;
106:     char* interface_id;
107:     var_ref* variables;
108:     qualifier_type* qualifiers;
109: } fault_msg;
110:
111: typedef struct data_message
112: {
113:     char* name;
114:     char* id;
115:     char* msg_arrival;
116:     char* description;
117:     char* msg_rate;
118:     char* interface_name;
119:     char* interface_id;
120:     var_ref* variables;
121:     qualifier_type* qualifiers;

```

```

122: struct data_message* next;
123: } data_msg;
124:
125: typedef struct command_message
126: {
127:     char* name;
128:     char* id;
129:     char* description;
130:     char* interface_name;
131:     char* interface_id;
132:     var_ref* variables;
133:     qualifier_type* qualifiers;
134:     struct command_message* next;
135: } cmd_msg;
136:
137: typedef struct command_type
138: {
139:     cmd_msg* command_message;
140:     fault_msg* fault_message;
141:     char* interface_name;
142:     char* interface_id;
143:     struct command_type* next;
144: }command;
145:
146: typedef struct notification_type
147: {
148:     data_msg* data_message;
149:     fault_msg* fault_message;
150:     char* interface_name;
151:     char* interface_id;
152:     struct notification_type* next;
153: }notification;
154:
155: typedef struct request_type
156: {
157:     cmd_msg* command_message;
158:     data_msg* data_message;
159:     fault_msg* fault_message;
160:     char* interface_name;
161:     char* interface_id;
162:     struct request_type* next;

```

```

163: }request;
164:
165: typedef struct message_type
166: {
167:     command* commands;
168:     notification* notifications;
169:     request* requests;
170: }message;
171:
172: typedef struct interface_type
173: {
174:     char* name;
175:     char* extends;
176:     char* id;
177:     char* description;
178:     qualifier_type* qualifiers;
179:     variable* variables;
180:     command* commands;
181:     notification* notifications;
182:     request* requests;
183:     struct interface_type* next;
184: }interface;
185:
186: typedef struct app_device_attributes
187: {
188:     char* name;
189:     char* kind;
190:     char* id;
191:     char* qualifier;
192:     char* description;
193:     char* manufacturer_id;
194:     char* componentKey;
195:     char* model_id;
196:     char* version_letter;
197:     char* version;
198:     char* serial_number;
199:     char* calibration_date;
200:     char* sensitivity_at_reference;
201:     char* reference_frequency;
202:     char* reference_temperature;
203:     char* measurement_range;

```

```

204: char* electrical_output;
205: char* quality_factor;
206: char* temperature_coefficient;
207: char* direction_xyz;
208: char* cal_due_date;
209: char* power_requirements;
210: char* spa_u_hub;
211: char* spa_u_port;
212: char* memory_minimum;
213: char* operating_system;
214: char* path_for_assembly;
215: char* path_on_spacecraft;
216: qualifier_type* qualifiers;
217: } app_dev_attr;
218:
219: typedef struct xteds
220: {
221:     char* name;
222:     char* version;
223:     char* description;
224:     char* xmlns;
225:     char* xmlns_xsi;
226:     char* schema_location;
227:     app_dev_attr* header;
228:     interface* interfaces;
229: } xteds;
230:
231: extern xteds* result;
232:
233: qualifier_type* link_qualifiers(qualifier_type*,qualifier_type*);
234: qualifier_type* merge_qualifiers(qualifier_type*,qualifier_type*);
235: qualifier_type* new_qualifier();
236:
237: coef* link_coefs(coef*,coef*);
238: coef* merge_coefs(coef*,coef*);
239: coef* new_coef();
240:
241: curve* merge_curves(curve*,curve*);
242: curve* new_curve();
243:
244: curveoption* link_options(curveoption*,curveoption*);

```



```

245: curveoption* merge_options(curveoption*,curveoption*);
246: curveoption* new_option();
247:
248: drange* merge_dranges(drange*,drange*);
249: drange* new_drange();
250:
251: location* merge_locations(location*,location*);
252: location* new_location();
253:
254: orientation* link_orientations(orientation*, orientation*);
255: orientation* merge_orientations(orientation*, orientation*);
256: orientation* new_orientation();
257:
258: variable* link_variables(variable*,variable*);
259: variable* merge_variables(variable*,variable*);
260: variable* new_variable();
261:
262: var_ref* link_variable_ref(var_ref*,var_ref*);
263: var_ref* new_variable_ref(char* name);
264:
265: fault_msg* merge_fault_msg(fault_msg*,fault_msg*);
266: fault_msg* new_fault_msg();
267: fault_msg* fault_add_var_refs(fault_msg*,var_ref*);
268:
269: data_msg* link_data_msg(data_msg*,data_msg*);
270: data_msg* merge_data_msg(data_msg*,data_msg*);
271: data_msg* new_data_msg();
272: data_msg* data_add_var_refs(data_msg*,var_ref*);
273:
274: cmd_msg* link_cmd_msg(cmd_msg*,cmd_msg*);
275: cmd_msg* merge_cmd_msg(cmd_msg*,cmd_msg*);
276: cmd_msg* new_cmd_msg();
277: cmd_msg* cmd_add_var_refs(cmd_msg*,var_ref*);
278:
279: command* new_command();
280: command* command_add_cmd_msg(command*,cmd_msg*);
281: command* command_add_fault_msg(command*,fault_msg*);
282: command* link_command(command*,command*);
283:
284: notification* new_notification();
285: notification* notification_add_data_msg(notification*,data_msg*);

```

```

286: notification* notification_add_fault_msg(notification*,fault_msg*);
287: notification* link_notification(notification*,notification*);
288:
289: request* new_request();
290: request* request_add_cmd_msg(request*,cmd_msg*);
291: request* request_add_data_msg(request*,data_msg*);
292: request* request_add_fault_msg(request*,fault_msg*);
293: request* link_request(request*,request*);
294:
295: message* new_message();
296: message* message_link_command(message*,command*);
297: message* message_link_notification(message*,notification*);
298: message* message_link_request(message*,request*);
299: message* merge_message(message*,message*);
300:
301: interface* new_interface();
302: interface* link_interface(interface*,interface*);
303: interface* merge_interface(interface*,interface*);
304: interface* interface_add_references(interface*,qualifier_type*,variable*,message*);
305:
306: xteds* merge_xteds(xteds*,xteds*);
307: xteds* new_xteds();
308: xteds* add_references(xteds*,app_dev_attr*,interface*);
309:
310: app_dev_attr* merge_app_dev_attr(app_dev_attr*,app_dev_attr*);
311: app_dev_attr* new_app_dev_attr();
312:
313: int delete_qualifier(qualifier_type*);
314: int delete_coef(coef*);
315: int delete_curve(curve*);
316: int delete_option(curveoption*);
317: int delete_drangle(drangle*);
318: int delete_location(location*);
319: int delete_orientation(orientation*);
320: int delete_variable(variable*);
321: int delete_var_ref(var_ref*);
322: int delete_fault_msg(fault_msg*);
323: int delete_data_msg(data_msg*);
324: int delete_cmd_msg(cmd_msg*);
325: int delete_command(command*);
326: int delete_notification(notification*);

```

```
327: int delete_request(request*);
328: int delete_message(message*);
329: int delete_interface(interface*);
330: extern SDMLIB_API int delete_xteds(xteds*);
331: int delete_app_dev_attr(app_dev_attr*);
332:
333: SDMLIB_API variable* getParsedVariable();
334: extern SDMLIB_API xteds* parsexTEDS(char*);
335: #endif
```

File: sdm/common/xTEDS/xTEDSQualifier.cpp

```
1: #include "xTEDSQualifier.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10: xTEDSQualifier::xTEDSQualifier(): m_strName(NULL), m_strDescription(NULL),
m_strUnits(NULL)
11: {
12: }
13:
14: xTEDSQualifier::xTEDSQualifier(char* pName, char* pDescription, char* pUnits):
m_strName(NULL), m_strDescription(NULL), m_strUnits(NULL)
15: {
16: setName(pName);
17: setDescription(pDescription);
18: setUnits(pUnits);
19: }
20:
21: xTEDSQualifier::xTEDSQualifier(const xTEDSQualifier& right): m_strName(NULL),
m_strDescription(NULL), m_strUnits(NULL)
22: {
23: if (right.m_strName != NULL)
24: m_strName = strdup(right.m_strName);
25: if (right.m_strDescription != NULL)
26: m_strDescription = strdup(right.m_strDescription);
27: if (right.m_strUnits != NULL)
28: m_strUnits = strdup(right.m_strUnits);
29: }
30:
31: xTEDSQualifier::~xTEDSQualifier()
32: {
33: if (m_strName != NULL)
34: free (m_strName);
35: if (m_strDescription != NULL)
36: free (m_strDescription);
37: if (m_strUnits != NULL)
```

```

38:     free (m_strUnits);
39: }
40:
41: void xTEDSQualifier::setName(const char* new_name)
42: {
43:     if (new_name == NULL) return;
44:     if (m_strName != NULL) free(m_strName);
45:     m_strName = strdup(new_name);
46: }
47:
48: void xTEDSQualifier::setDescription(const char* new_description)
49: {
50:     if (new_description == NULL) return;
51:     if (m_strDescription != NULL) free(m_strDescription);
52:     m_strDescription = strdup(new_description);
53: }
54:
55: void xTEDSQualifier::setUnits(const char* new_units)
56: {
57:     if (new_units == NULL) return;
58:     if (m_strUnits != NULL) free(m_strUnits);
59:     m_strUnits = strdup(new_units);
60: }
61:
62: const char* xTEDSQualifier::getName(void) const
63: {
64:     return m_strName;
65: }
66:
67: const char* xTEDSQualifier::getDescription(void) const
68: {
69:     return m_strDescription;
70: }
71:
72: const char* xTEDSQualifier::getUnits(void) const
73: {
74:     return m_strUnits;
75: }
76:
77: bool xTEDSQualifier::MatchesName(const char* MatchString) const
78: {

```

```

79: if (MatchString != NULL && strcmp(MatchString, m_strName)==0)
80:     return true;
81: return false;
82: }
83:
84: bool xTEDSQualifier::MatchesDescription(const char* MatchString) const
85: {
86: if (MatchString != NULL && strcmp(MatchString, m_strDescription)==0)
87:     return true;
88: return false;
89: }
90:
91: /*xTEDSQualifier& xTEDSQualifier::operator=(const xTEDSQualifier& b)
92: {
93: if(name!=NULL) free(name);
94: name = strdup(b.name);
95: if(description!=NULL) free(description);
96: description = strdup(b.description);
97: return *this;
98: }
99: xTEDSQualifier::xTEDSQualifier(const xTEDSQualifier& b)
100: {
101:     m_strName[0] = m_strDescription[0] = m_strUnits[0] = '\0';
102:     setName(b.m_strName);
103:     setDescription(b.m_strDescription);
104:     setUnits(b.m_strUnits);
105: }*/
106:
107: bool xTEDSQualifier::Parse(const char* quals)
108: {
109:     char temp[128];
110:     size_t start = 0;
111:     size_t count = 0;
112:     size_t end = 0;
113:
114:     int i;
115:     bool valid = false;
116:
117:     count = strlen(quals);
118:     i=0;
119:     while(quals[i] != ' ' && quals[i] != '=' && quals[i] != '\0')

```

```

120:  {
121:      i++;
122:  }
123:  if(quals[i] != '\0')
124:      valid = true;
125:  end = i;
126:  if(valid)
127:  {
128:      strncpy(temp,quals,end);
129:      temp[end] = 0;
130:      setName(temp);
131:  }
132:  start = end;
133:  end = count;
134:
135:  while(quals[start] != "" && quals[start] != '\0')
136:  {
137:      start++;
138:  }
139:  if(quals[start] == '\0')
140:      valid = false;
141:  else
142:      start += 1;
143:  while(quals[end] != "" && end > start)
144:  {
145:      end--;
146:  }
147:  if(end == start)
148:      valid = false;
149:  if(valid)
150:  {
151:      strncpy(temp,&quals[start],end-start);
152:      temp[end-start] = 0;
153:      setDescription(temp);
154:  }
155:  return valid;
156: }
157:
158: void xTEDSQualifier::QualifierInfoRequest( char* InfoBufferOut, size_t BufferSize )
159: {
160:     char Buf[MSG_DEF_SIZE], TempBuf[512];

```

```

161:   size_t BufLength = 0;
162:   if (m_strName == NULL || m_strDescription == NULL)
163:       return ;
164:
165:   BufLength = snprintf(Buf, sizeof(Buf), "<Qualifier name= \"%s \" value= \"%s \"", m_strName,
m_strDescription);
166:
167:   // Add units, if given
168:   if(m_strUnits != NULL)
169:   {
170:       snprintf(TempBuf, sizeof(TempBuf), " units= \"%s \"", m_strUnits);
171:
172:       strncat(Buf, TempBuf, sizeof(Buf) - BufLength);
173:       BufLength = strlen(Buf);
174:   }
175:   // Close off the tag
176:   strncat(Buf, " />", sizeof(Buf) - BufLength);
177:
178:   strncat(InfoBufferOut, Buf, BufferSize - 1);
179: }

```


File: sdm/common/xTEDS/xTEDSSubscription.h

```
1: #ifndef _SDM_XTEDS_SUBSCRIPTION_H_  
2: #define _SDM_XTEDS_SUBSCRIPTION_H_  
3:  
4:  
5:  
6:  
7: #endif
```

File: sdm/common/xTEDS/xTEDSLocation.cpp

```
1: #include <string.h>
2: #include <stdlib.h>
3: #include <stdio.h>
4: #include "xTEDSLocation.h"
5: #include "../message_defs.h"
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10:
11: xTEDSLocation::xTEDSLocation():m_strX(NULL),m_strY(NULL),m_strZ(NULL),m_strUnits(NULL)
12: {
13: }
14:
15: xTEDSLocation::~xTEDSLocation()
16: {
17:     if (m_strX != NULL) free(m_strX);
18:     if (m_strY != NULL) free(m_strY);
19:     if (m_strZ != NULL) free(m_strZ);
20:     if (m_strUnits != NULL) free(m_strUnits);
21: }
22:
23: void xTEDSLocation::setX(const char *xStr)
24: {
25:     if (xStr == NULL) return;
26:     if (m_strX != NULL) free(m_strX);
27:     m_strX = strdup(xStr);
28: }
29:
30: void xTEDSLocation::setY(const char *yStr)
31: {
32:     if (yStr == NULL) return;
33:     if (m_strY != NULL) free(m_strY);
34:     m_strY = strdup(yStr);
35: }
36:
37: void xTEDSLocation::setZ(const char *zStr)
38: {
39:     if (zStr == NULL) return;
```

```

39: if (m_strZ != NULL) free(m_strZ);
40: m_strZ = strdup(zStr);
41: }
42:
43: void xTEDSLocation::setUnits(const char *unitsStr)
44: {
45: if (unitsStr == NULL) return;
46: if (m_strUnits != NULL) free(m_strUnits);
47: m_strUnits = strdup(unitsStr);
48: }
49:
50: void xTEDSLocation::setLocation(const location *loc)
51: {
52: if (loc == NULL) return;
53: setX(loc->x_value);
54: setY(loc->y_value);
55: setZ(loc->z_value);
56: setUnits(loc->units);
57: }
58:
59: void xTEDSLocation::setLocation(const char *x, const char *y, const char *z, const char *units)
60: {
61: setX(x);
62: setY(y);
63: setZ(z);
64: setUnits(units);
65: }
66:
67: void xTEDSLocation::VarInfoRequest( char* InfoBufferOut, size_t BufferSize )
68: {
69: char Buf[MSG_DEF_SIZE];
70:
71: snprintf(Buf, sizeof(Buf), "<Location x= \"%s \" y= \"%s \" z= \"%s \" units= \"%s \" />", m_strX,
m_strY, m_strZ, m_strUnits);
72:
73: strncat(InfoBufferOut, Buf, BufferSize - 1);
74: }

```

File: sdm/common/xTEDS/xTEDSQualifierList.h

```
1: #ifndef __SDM_XTEDS_QUALIFIER_LIST_H_
2: #define __SDM_XTEDS_QUALIFIER_LIST_H_
3:
4: //The xTEDSQualifierList does not 'own' its items, it is just a way for items to reference other items in
the tree
5: #include "../message_defs.h"
6: #include "xTEDSQualifier.h"
7: #include <stdio.h>
8: #include <cstring>
9:
10: struct xTEDSQualifierListNode
11: {
12: xTEDSQualifier* data;
13: struct xTEDSQualifierListNode* next;
14: };
15:
16: class xTEDSQualifierList
17: {
18: public:
19: xTEDSQualifierList();
20: xTEDSQualifierList(const xTEDSQualifierList&);
21: ~xTEDSQualifierList();
22: void addQualifier(const xTEDSQualifier &NewQualifier);
23: void Parse(const char*);
24: xTEDSQualifierList& operator=(const xTEDSQualifierList&);
25: void QualifierInfoRequest(char* InfoBufferOut, size_t BufferSize);
26:
27: int Size() const;
28: bool isEmpty() const;
29: const xTEDSQualifier& operator[] (int index) const;
30: private:
31: int itemCount;
32: struct xTEDSQualifierListNode* head;
33: struct xTEDSQualifierListNode* tail;
34: };
35:
36: #endif
```

File: sdm/common/xTEDS/xTEDSDrange.cpp

```
1: #include "xTEDSDrange.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10: xTEDSDrange::xTEDSDrange():m_strName(NULL),m_strDescription(NULL),m_xolOptions()
11: {
12: }
13:
14: /* Not used..
15:
16: xTEDSDrange::xTEDSDrange(const xTEDSDrange&
17: b):m_strName(NULL),m_strDescription(NULL),m_xolOptions()
18: {
19: if(b.m_strName!=NULL) free(m_strName);
20: m_strName = strdup(b.m_strName);
21: if(b.m_strDescription!=NULL) free(m_strDescription);
22: m_strDescription = strdup(b.m_strDescription);
23: //TODO: Option list not copied
24: }
25:
26: xTEDSDrange& xTEDSDrange::operator=(const xTEDSDrange& b)
27: {
28: if(m_strName!=NULL) free(m_strName);
29: m_strName = strdup(b.m_strName);
30: if(m_strDescription!=NULL) free(m_strDescription);
31: m_strDescription = strdup(b.m_strDescription);
32: //TODO: Option list not copied
33: return *this;
34: }*/
35:
36:
37:
38: xTEDSDrange::~xTEDSDrange()
39: {
40: if(m_strName!=NULL) free(m_strName);
41: if(m_strDescription!=NULL) free(m_strDescription);
42: }
```

```

39:
40: void xTEDSDrange::setName(const char* new_name)
41: {
42: if (new_name == NULL) return;
43: if(m_strName!=NULL) free(m_strName);
44: m_strName = strdup(new_name);
45: }
46:
47: void xTEDSDrange::setDescription(const char* new_description)
48: {
49: if (new_description == NULL) return;
50: if(m_strDescription!=NULL) free(m_strDescription);
51: m_strDescription = strdup(new_description);
52: }
53:
54: void xTEDSDrange::setDRange(const drange* NewRange)
55: {
56: setName(NewRange->name);
57: setDescription(NewRange->description);
58: m_xolOptions.setOptionsList(NewRange->options);
59: }
60:
61: void xTEDSDrange::VarInfoRequest( char* InfoBufferOut, size_t BufferSize ) const
62: {
63: char Buf[MSG_DEF_SIZE], TempBuf[512];
64:
65: if (m_strName == NULL)
66:     return ;
67:
68: snprintf(Buf, sizeof(Buf), "<Drange name= \"%s \", m_strName);
69:
70: if (m_strDescription != NULL)
71: {
72:     snprintf(TempBuf, sizeof(TempBuf), " description= \"%s \", m_strDescription);
73:
74:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
75: }
76:
77: strncat(Buf, ">", sizeof(Buf) - strlen(Buf));
78:
79: size_t CurBufLength = strlen(Buf);

```

```
80: m_xolOptions.VarInfoRequest(Buf + CurBufLength, sizeof(Buf) - CurBufLength);
81:
82: strncat(Buf, "\n \t</Drange>", sizeof(Buf) - strlen(Buf));
83:
84: strncat(InfoBufferOut, Buf, BufferSize - 1);
85: }
```

File: sdm/common/xTEDS/xTEDSCoef.h

```
1: #ifndef __SDM_XTEDS_COEF_H_
2: #define __SDM_XTEDS_COEF_H_
3:
4: #include "../message_defs.h"
5: extern "C"
6: {
7: #include "xTEDSParser.h"
8: }
9: #include <cstring>
10:
11: class xTEDSCoef
12: {
13: public:
14: xTEDSCoef();
15: xTEDSCoef(const xTEDSCoef&);
16: xTEDSCoef& operator=(const xTEDSCoef&);
17: ~xTEDSCoef();
18:
19: void setCoef(const coef* NewCoef);
20: void setDescription(const char*);
21: void setValue(const char*);
22: void setExponent(const char*);
23:
24: void VarInfoRequest(char* InfoBufferOut, size_t BufferSize) const;
25: private:
26: char* m_strDescription;
27: char* m_strValue;
28: char* m_strExponent;
29: };
30:
31: #endif
```


File: sdm/common/xTEDS/lex.xTEDS.c

```
1:
2: #line 3 "lex.xTEDS.c"
3:
4: #define YY_INT_ALIGNED short int
5:
6: /* A lexical scanner generated by flex */
7:
8: #define FLEX_SCANNER
9: #define YY_FLEX_MAJOR_VERSION 2
10: #define YY_FLEX_MINOR_VERSION 5
11: #define YY_FLEX_SUBMINOR_VERSION 33
12: #if YY_FLEX_SUBMINOR_VERSION > 0
13: #define FLEX_BETA
14: #endif
15:
16: /* First, we deal with platform-specific or compiler-specific issues. */
17:
18: /* begin standard C headers. */
19: #include <stdio.h>
20: #include <string.h>
21: #include <errno.h>
22: #include <stdlib.h>
23:
24: /* end standard C headers. */
25:
26: /* flex integer type definitions */
27:
28: #ifndef FLEXINT_H
29: #define FLEXINT_H
30:
31: /* C99 systems have <inttypes.h>. Non-C99 systems may or may not. */
32:
33: #if __STDC_VERSION__ >= 199901L
34:
35: /* C99 says to define __STDC_LIMIT_MACROS before including stdint.h,
36:  * if you want the limit (max/min) macros for int types.
37:  */
38: #ifndef __STDC_LIMIT_MACROS
39: #define __STDC_LIMIT_MACROS 1
```

```

40: #endif
41:
42: #include <inttypes.h>
43: typedef int8_t flex_int8_t;
44: typedef uint8_t flex_uint8_t;
45: typedef int16_t flex_int16_t;
46: typedef uint16_t flex_uint16_t;
47: typedef int32_t flex_int32_t;
48: typedef uint32_t flex_uint32_t;
49: #else
50: typedef signed char flex_int8_t;
51: typedef short int flex_int16_t;
52: typedef int flex_int32_t;
53: typedef unsigned char flex_uint8_t;
54: typedef unsigned short int flex_uint16_t;
55: typedef unsigned int flex_uint32_t;
56: #endif /* ! C99 */
57:
58: /* Limits of integral types. */
59: #ifndef INT8_MIN
60: #define INT8_MIN      (-128)
61: #endif
62: #ifndef INT16_MIN
63: #define INT16_MIN     (-32767-1)
64: #endif
65: #ifndef INT32_MIN
66: #define INT32_MIN     (-2147483647-1)
67: #endif
68: #ifndef INT8_MAX
69: #define INT8_MAX      (127)
70: #endif
71: #ifndef INT16_MAX
72: #define INT16_MAX     (32767)
73: #endif
74: #ifndef INT32_MAX
75: #define INT32_MAX     (2147483647)
76: #endif
77: #ifndef UINT8_MAX
78: #define UINT8_MAX     (255U)
79: #endif
80: #ifndef UINT16_MAX

```

```

81: #define UINT16_MAX      (65535U)
82: #endif
83: #ifndef UINT32_MAX
84: #define UINT32_MAX      (4294967295U)
85: #endif
86:
87: #endif /* ! FLEXINT_H */
88:
89: #ifdef __cplusplus
90:
91: /* The "const" storage-class-modifier is valid. */
92: #define YY_USE_CONST
93:
94: #else /* ! __cplusplus */
95:
96: #if __STDC__
97:
98: #define YY_USE_CONST
99:
100: #endif /* __STDC__ */
101: #endif /* ! __cplusplus */
102:
103: #ifdef YY_USE_CONST
104: #define yyconst const
105: #else
106: #define yyconst
107: #endif
108:
109: /* Returned upon end-of-file. */
110: #define YY_NULL 0
111:
112: /* Promotes a possibly negative, possibly signed char to an unsigned
113:  * integer for use as an array index.  If the signed char is negative,
114:  * we want to instead treat it as an 8-bit unsigned char, hence the
115:  * double cast.
116:  */
117: #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
118:
119: /* Enter a start condition.  This macro really ought to take a parameter,
120:  * but we do it the disgusting crufty way forced on us by the ()-less
121:  * definition of BEGIN.

```

```

122: */
123: #define BEGIN (yy_start) = 1 + 2 *
124:
125: /* Translate the current start state into a value that can be later handed
126: * to BEGIN to return to the state. The YYSTATE alias is for lex
127: * compatibility.
128: */
129: #define YY_START (((yy_start) - 1) / 2)
130: #define YYSTATE YY_START
131:
132: /* Action number for EOF rule of a given start state. */
133: #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
134:
135: /* Special action meaning "start processing a new file". */
136: #define YY_NEW_FILE xTEDSrestart(xTEDSin )
137:
138: #define YY_END_OF_BUFFER_CHAR 0
139:
140: /* Size of default input buffer. */
141: #ifndef YY_BUF_SIZE
142: #define YY_BUF_SIZE 16384
143: #endif
144:
145: /* The state buf must be large enough to hold one state per character in the main buffer.
146: */
147: #define YY_STATE_BUF_SIZE ((YY_BUF_SIZE + 2) * sizeof(yy_state_type))
148:
149: #ifndef YY_TYPEDEF_YY_BUFFER_STATE
150: #define YY_TYPEDEF_YY_BUFFER_STATE
151: typedef struct yy_buffer_state *YY_BUFFER_STATE;
152: #endif
153:
154: extern int xTEDSleng;
155:
156: extern FILE *xTEDSin, *xTEDSout;
157:
158: #define EOB_ACT_CONTINUE_SCAN 0
159: #define EOB_ACT_END_OF_FILE 1
160: #define EOB_ACT_LAST_MATCH 2
161:
162: #define YY_LESS_LINENO(n)

```

```

163:
164: /* Return all but the first "n" matched characters back to the input stream. */
165: #define yyless(n) \
166:     do \
167:         { \
168:             /* Undo effects of setting up xTEDStext. */ \
169:             int yyless_macro_arg = (n); \
170:             YY_LESS_LINENO(yyless_macro_arg); \
171:             *yy_cp = (yy_hold_char); \
172:             YY_RESTORE_YY_MORE_OFFSET \
173:             (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ; \
174:             YY_DO_BEFORE_ACTION; /* set up xTEDStext again */ \
175:         } \
176:     while ( 0 )
177:
178: #define unput(c) yyunput( c, (yytext_ptr) )
179:
180: /* The following is because we cannot portably get our hands on size_t
181:  * (without autoconf's help, which isn't available because we want
182:  * flex-generated scanners to compile on their own).
183:  */
184:
185: #ifndef YY_TYPEDEF_YY_SIZE_T
186: #define YY_TYPEDEF_YY_SIZE_T
187: typedef unsigned int yy_size_t;
188: #endif
189:
190: #ifndef YY_STRUCT_YY_BUFFER_STATE
191: #define YY_STRUCT_YY_BUFFER_STATE
192: struct yy_buffer_state
193: {
194:     FILE *yy_input_file;
195:
196:     char *yy_ch_buf;      /* input buffer */
197:     char *yy_buf_pos;     /* current position in input buffer */
198:
199:     /* Size of input buffer in bytes, not including room for EOB
200:      * characters.
201:      */
202:     yy_size_t yy_buf_size;
203:

```

```

204:  /* Number of characters read into yy_ch_buf, not including EOB
205:   * characters.
206:   */
207:  int yy_n_chars;
208:
209:  /* Whether we "own" the buffer - i.e., we know we created it,
210:   * and can realloc() it to grow it, and should free() it to
211:   * delete it.
212:   */
213:  int yy_is_our_buffer;
214:
215:  /* Whether this is an "interactive" input source; if so, and
216:   * if we're using stdio for input, then we want to use getc()
217:   * instead of fread(), to make sure we stop fetching input after
218:   * each newline.
219:   */
220:  int yy_is_interactive;
221:
222:  /* Whether we're considered to be at the beginning of a line.
223:   * If so, '^' rules will be active on the next match, otherwise
224:   * not.
225:   */
226:  int yy_at_bol;
227:
228:  int yy_bs_lineno; /**< The line count. */
229:  int yy_bs_column; /**< The column count. */
230:
231:  /* Whether to try to fill the input buffer when we reach the
232:   * end of it.
233:   */
234:  int yy_fill_buffer;
235:
236:  int yy_buffer_status;
237:
238: #define YY_BUFFER_NEW 0
239: #define YY_BUFFER_NORMAL 1
240:  /* When an EOF's been seen but there's still some text to process
241:   * then we mark the buffer as YY_EOF_PENDING, to indicate that we
242:   * shouldn't try reading from the input source any more.  We might
243:   * still have a bunch of tokens to match, though, because of
244:   * possible backing-up.

```

```

245:  *
246:  * When we actually see the EOF, we change the status to "new"
247:  * (via xTEDSrestart()), so that the user can continue scanning by
248:  * just pointing xTEDSin at a new input file.
249:  */
250: #define YY_BUFFER_EOF_PENDING 2
251:
252: };
253: #endif /* !YY_STRUCT_YY_BUFFER_STATE */
254:
255: /* Stack of input buffers. */
256: static size_t yy_buffer_stack_top = 0; /**< index of top of stack. */
257: static size_t yy_buffer_stack_max = 0; /**< capacity of stack. */
258: static YY_BUFFER_STATE * yy_buffer_stack = 0; /**< Stack as an array. */
259:
260: /* We provide macros for accessing buffer states in case in the
261:  * future we want to put the buffer states in a more general
262:  * "scanner state".
263:  *
264:  * Returns the top of the stack, or NULL.
265:  */
266: #define YY_CURRENT_BUFFER ( (yy_buffer_stack) \
267:                             ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
268:                             : NULL)
269:
270: /* Same as previous macro, but useful when we know that the buffer stack is not
271:  * NULL or when we need an lvalue. For internal use only.
272:  */
273: #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
274:
275: /* yy_hold_char holds the character lost when xTEDStext is formed. */
276: static char yy_hold_char;
277: static int yy_n_chars;      /* number of characters read into yy_ch_buf */
278: int xTEDSleng;
279:
280: /* Points to current character in buffer. */
281: static char *yy_c_buf_p = (char *) 0;
282: static int yy_init = 0;     /* whether we need to initialize */
283: static int yy_start = 0;    /* start state number */
284:
285: /* Flag which is used to allow xTEDSwrap()'s to do buffer switches

```

```

286: * instead of setting up a fresh xTEDSin. A bit of a hack ...
287: */
288: static int yy_did_buffer_switch_on_eof;
289:
290: void xTEDSrestart (FILE *input_file );
291: void xTEDS_switch_to_buffer (YY_BUFFER_STATE new_buffer );
292: YY_BUFFER_STATE xTEDS_create_buffer (FILE *file,int size );
293: void xTEDS_delete_buffer (YY_BUFFER_STATE b );
294: void xTEDS_flush_buffer (YY_BUFFER_STATE b );
295: void xTEDSpush_buffer_state (YY_BUFFER_STATE new_buffer );
296: void xTEDSpop_buffer_state (void );
297:
298: static void xTEDSensure_buffer_stack (void );
299: static void xTEDS_load_buffer_state (void );
300: static void xTEDS_init_buffer (YY_BUFFER_STATE b,FILE *file );
301:
302: #define YY_FLUSH_BUFFER xTEDS_flush_buffer(YY_CURRENT_BUFFER )
303:
304: YY_BUFFER_STATE xTEDS_scan_buffer (char *base,yy_size_t size );
305: YY_BUFFER_STATE xTEDS_scan_string (yyconst char *yy_str );
306: YY_BUFFER_STATE xTEDS_scan_bytes (yyconst char *bytes,int len );
307:
308: void *xTEDSalloc (yy_size_t );
309: void *xTEDSrealloc (void *,yy_size_t );
310: void xTEDSfree (void * );
311:
312: #define yy_new_buffer xTEDS_create_buffer
313:
314: #define yy_set_interactive(is_interactive) \
315:   { \
316:     if ( ! YY_CURRENT_BUFFER ){ \
317:       xTEDSensure_buffer_stack (); \
318:       YY_CURRENT_BUFFER_LVALUE = \
319:         xTEDS_create_buffer(xTEDSin,YY_BUF_SIZE ); \
320:     } \
321:     YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
322:   }
323:
324: #define yy_set_bol(at_bol) \
325:   { \
326:     if ( ! YY_CURRENT_BUFFER ){ \

```



```

327:     xTEDSensure_buffer_stack (); \
328:     YY_CURRENT_BUFFER_LVALUE = \
329:     xTEDS_create_buffer(xTEDSin,YY_BUF_SIZE ); \
330: } \
331: YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
332: }
333:
334: #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
335:
336: /* Begin user sect3 */
337:
338: typedef unsigned char YY_CHAR;
339:
340: FILE *xTEDSin = (FILE *) 0, *xTEDSout = (FILE *) 0;
341:
342: typedef int yy_state_type;
343:
344: extern int xTEDSlineno;
345:
346: int xTEDSlineno = 1;
347:
348: extern char *xTEDStext;
349: #define yytext_ptr xTEDStext
350:
351: static yy_state_type yy_get_previous_state (void );
352: static yy_state_type yy_try_NUL_trans (yy_state_type current_state );
353: static int yy_get_next_buffer (void );
354: static void yy_fatal_error (yyconst char msg[] );
355:
356: /* Done after the current pattern has been matched and before the
357:  * corresponding action - sets up xTEDStext.
358:  */
359: #define YY_DO_BEFORE_ACTION \
360:     (yytext_ptr) = yy_bp; \
361:     xTEDSleng = (size_t) (yy_cp - yy_bp); \
362:     (yy_hold_char) = *yy_cp; \
363:     *yy_cp = '\0'; \
364:     (yy_c_buf_p) = yy_cp;
365:
366: #define YY_NUM_RULES 111
367: #define YY_END_OF_BUFFER 112

```

```

368: /* This struct is not used in this scanner,
369:  but its presence is necessary. */
370: struct yy_trans_info
371: {
372:     flex_int32_t yy_verify;
373:     flex_int32_t yy_nxt;
374: };
375: static yyconst flex_int16_t yy_accept[815] =
376: { 0,
377:   108, 108, 112, 110, 108, 109, 110, 110, 110, 106,
378:   110,  1,  2, 110, 110, 110, 110, 110, 110, 110,
379:   110, 110, 110, 110, 110, 110, 110, 110, 110, 110,
380:   110, 110, 110, 90, 91, 92,  0, 108,  0, 104,
381:   0, 105,  3,  0, 106,  0,  0,  0,  0,  0,
382:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
383:   4,  0,  0,  0,  0,  0,  0,  0,  0,  0,
384:   0,  0,  0,  0, 48,  0,  0,  0,  0,  0,
385:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
386:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
387:
388:   0,  0,  0,  0, 45, 45,  0,  0,  0,  0,
389:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
390:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
391:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
392:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
393:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
394:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
395:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
396:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
397:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
398:
399:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
400:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
401:   0, 93,  0,  0,  0,  0,  0,  0,  0,  0,
402:   0,  0,  0,  0, 47,  0,  0,  0,  0,  0,
403:   0,  0, 46,  0,  0,  0,  0,  0,  0, 97,
404:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
405:   0,  0,  0,  0, 99,  0,  0,  0,  0,  0,
406:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
407:   0,  0,  0,  0,  0,  0,  5,  0, 18,  0,
408:   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,

```

409:
 410: 0, 0, 0, 0, 0, 0, 68, 94, 0, 0,
 411: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 412: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 413: 0, 0, 0, 98, 0, 0, 0, 0, 0, 0,
 414: 0, 0, 0, 77, 67, 0, 101, 0, 100, 0,
 415: 0, 0, 0, 0, 0, 0, 19, 0, 0, 0,
 416: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 417: 0, 0, 0, 0, 16, 0, 0, 0, 0, 0,
 418: 0, 0, 0, 0, 0, 0, 0, 0, 7, 0,
 419: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 420:
 421: 0, 73, 0, 80, 0, 0, 0, 0, 0, 0,
 422: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 423: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 424: 0, 107, 0, 0, 17, 0, 0, 0, 0, 0,
 425: 0, 0, 0, 0, 0, 0, 0, 0, 6, 0,
 426: 0, 0, 0, 26, 12, 0, 0, 0, 0, 14,
 427: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 428: 0, 0, 0, 0, 0, 0, 85, 0, 0, 0,
 429: 0, 53, 0, 70, 0, 0, 0, 0, 0, 0,
 430: 0, 0, 0, 0, 0, 0, 0, 0, 83, 0,
 431:
 432: 0, 0, 52, 0, 0, 0, 0, 0, 0, 27,
 433: 13, 0, 0, 0, 0, 15, 0, 0, 0, 0,
 434: 0, 30, 20, 0, 0, 0, 0, 0, 0, 0,
 435: 34, 0, 0, 0, 74, 0, 0, 0, 0, 0,
 436: 0, 0, 95, 81, 0, 0, 0, 0, 0, 0,
 437: 0, 0, 0, 0, 0, 0, 72, 75, 0, 0,
 438: 0, 0, 0, 84, 0, 0, 0, 0, 0, 0,
 439: 31, 21, 0, 0, 0, 0, 0, 0, 0, 35,
 440: 0, 0, 0, 0, 36, 0, 40, 0, 0, 0,
 441: 10, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 442:
 443: 0, 0, 0, 0, 0, 0, 0, 0, 71, 49,
 444: 0, 0, 0, 0, 0, 0, 0, 0, 0, 102,
 445: 0, 0, 0, 0, 37, 0, 41, 0, 0, 0,
 446: 11, 0, 0, 0, 28, 0, 0, 38, 0, 65,
 447: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 448: 69, 0, 0, 0, 0, 0, 0, 0, 0, 79,
 449: 0, 0, 96, 0, 0, 0, 0, 0, 0, 29,

```

450:    0, 0, 39, 0, 0, 24, 0, 0, 0, 0,
451:    0, 0, 0, 50, 0, 0, 0, 0, 0, 0,
452:    0, 0, 0, 0, 0, 0, 0, 0, 76, 0, 0,
453:
454:    0, 0, 0, 0, 25, 0, 0, 0, 0, 8,
455:    0, 0, 42, 22, 0, 82, 78, 64, 0, 44,
456:    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
457:    0, 55, 0, 0, 0, 9, 0, 0, 43, 23,
458:    32, 0, 0, 0, 0, 86, 0, 0, 0, 0,
459:    62, 0, 0, 0, 0, 54, 0, 33, 0, 0,
460:    51, 0, 0, 0, 0, 0, 0, 0, 0, 0,
461:    0, 56, 0, 0, 87, 88, 0, 0, 0, 0,
462:    0, 0, 0, 61, 60, 89, 0, 0, 0, 0,
463:    0, 0, 66, 0, 0, 0, 0, 0, 58, 0,
464:
465:    0, 0, 103, 0, 0, 0, 59, 0, 0, 0,
466:    0, 57, 63, 0
467:  };
468:
469: static yyconst flex_int32_t yy_ec[256] =
470:  { 0,
471:    1, 1, 1, 1, 1, 1, 1, 1, 2, 3,
472:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
473:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
474:    1, 4, 5, 6, 1, 1, 1, 1, 1, 1,
475:    1, 1, 1, 1, 7, 8, 9, 10, 10, 10,
476:    10, 10, 10, 10, 10, 10, 10, 11, 1, 12,
477:    13, 14, 15, 1, 16, 1, 17, 18, 19, 20,
478:    1, 21, 22, 1, 23, 24, 25, 26, 27, 28,
479:    29, 30, 31, 32, 33, 34, 1, 35, 36, 37,
480:    1, 1, 1, 1, 38, 1, 39, 40, 41, 42,
481:
482:    43, 44, 45, 46, 47, 1, 48, 49, 50, 51,
483:    52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
484:    62, 63, 1, 1, 1, 1, 1, 1, 1, 1,
485:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
486:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
487:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
488:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
489:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
490:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```

```

491:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
492:
493:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
494:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
495:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
496:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
497:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
498:    1, 1, 1, 1, 1
499: };
500:
501: static yyconst flex_int32_t yy_meta[64] =
502: { 0,
503:    1, 2, 2, 2, 1, 1, 1, 1, 1, 1,
504:    1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
505:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
506:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
507:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
508:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
509:    1, 1, 1
510: };
511:
512: static yyconst flex_int16_t yy_base[820] =
513: { 0,
514:    0, 0, 900, 852, 62, 901, 59, 57, 54, 61,
515:    104, 851, 850, 56, 44, 32, 35, 33, 37, 38,
516:    60, 849, 49, 71, 50, 47, 90, 58, 111, 100,
517:    52, 89, 105, 99, 135, 848, 852, 73, 887, 901,
518:    119, 882, 901, 881, 107, 883, 167, 828, 835, 108,
519:    124, 848, 835, 833, 832, 109, 825, 839, 842, 848,
520:    901, 863, 837, 838, 831, 828, 825, 823, 124, 131,
521:    829, 830, 117, 815, 0, 810, 127, 817, 816, 138,
522:    824, 820, 814, 820, 805, 801, 817, 820, 811, 805,
523:    805, 811, 815, 120, 814, 813, 801, 803, 800, 793,
524:
525:    798, 799, 798, 792, 901, 837, 835, 788, 140, 156,
526:    801, 788, 786, 785, 147, 778, 792, 795, 801, 782,
527:    778, 160, 775, 772, 769, 788, 768, 768, 783, 766,
528:    765, 774, 781, 765, 763, 798, 778, 757, 759, 764,
529:    756, 158, 758, 771, 768, 765, 766, 754, 753, 761,
530:    753, 763, 759, 755, 741, 742, 745, 753, 174, 752,
531:    739, 747, 749, 750, 741, 744, 728, 742, 743, 736,

```

532: 728, 736, 749, 730, 727, 722, 720, 721, 725, 764,
 533: 729, 713, 205, 719, 163, 716, 713, 710, 729, 709,
 534: 709, 724, 707, 706, 715, 722, 706, 704, 739, 708,
 535:
 536: 707, 711, 704, 694, 713, 704, 699, 700, 705, 708,
 537: 699, 698, 701, 694, 684, 694, 722, 706, 683, 687,
 538: 693, 901, 677, 694, 681, 674, 676, 689, 672, 686,
 539: 676, 675, 686, 675, 901, 666, 678, 663, 665, 670,
 540: 663, 678, 901, 677, 187, 660, 667, 666, 666, 901,
 541: 668, 655, 666, 661, 668, 194, 664, 662, 648, 660,
 542: 655, 645, 644, 653, 901, 213, 214, 216, 691, 648,
 543: 652, 645, 635, 654, 645, 640, 641, 646, 649, 640,
 544: 639, 642, 635, 625, 635, 663, 901, 633, 901, 640,
 545: 635, 199, 636, 631, 618, 619, 616, 628, 619, 619,
 546:
 547: 622, 625, 628, 635, 627, 625, 901, 901, 620, 607,
 548: 610, 611, 612, 601, 602, 609, 612, 612, 596, 605,
 549: 605, 611, 594, 586, 625, 591, 588, 587, 591, 591,
 550: 611, 584, 174, 901, 614, 595, 205, 580, 587, 577,
 551: 582, 594, 577, 901, 901, 579, 619, 588, 901, 223,
 552: 229, 621, 620, 230, 612, 578, 901, 585, 580, 209,
 553: 581, 576, 563, 564, 561, 573, 564, 564, 567, 570,
 554: 573, 580, 569, 558, 901, 552, 564, 563, 562, 579,
 555: 559, 555, 554, 549, 542, 554, 541, 556, 901, 189,
 556: 554, 576, 554, 549, 534, 537, 542, 541, 536, 535,
 557:
 558: 529, 901, 542, 901, 542, 539, 556, 538, 532, 535,
 559: 530, 521, 544, 531, 526, 525, 509, 201, 519, 530,
 560: 517, 520, 540, 525, 509, 514, 523, 510, 499, 513,
 561: 544, 901, 516, 505, 901, 499, 511, 510, 509, 526,
 562: 506, 502, 501, 496, 489, 501, 488, 503, 901, 503,
 563: 499, 495, 486, 901, 901, 482, 498, 484, 494, 901,
 564: 495, 486, 475, 482, 472, 477, 466, 488, 469, 474,
 565: 490, 466, 470, 480, 475, 462, 901, 484, 460, 466,
 566: 468, 901, 455, 901, 462, 496, 458, 456, 457, 465,
 567: 487, 445, 454, 463, 462, 455, 442, 442, 901, 442,
 568:
 569: 446, 440, 472, 439, 451, 454, 450, 446, 437, 901,
 570: 901, 433, 449, 435, 445, 901, 446, 437, 426, 433,
 571: 424, 455, 901, 430, 433, 436, 425, 436, 417, 430,
 572: 901, 429, 431, 415, 901, 412, 421, 410, 427, 418,

```

573: 413, 424, 901, 901, 423, 403, 417, 408, 419, 412,
574: 400, 416, 396, 402, 397, 412, 901, 901, 407, 392,
575: 391, 400, 396, 901, 394, 386, 400, 395, 391, 383,
576: 414, 901, 389, 392, 395, 384, 395, 376, 389, 901,
577: 388, 383, 373, 366, 901, 384, 901, 369, 378, 369,
578: 393, 365, 378, 368, 396, 369, 365, 381, 366, 365,
579:
580: 358, 361, 364, 361, 378, 352, 366, 359, 901, 901,
581: 364, 221, 352, 347, 345, 38, 69, 84, 115, 901,
582: 142, 180, 187, 182, 901, 202, 901, 189, 200, 194,
583: 220, 199, 207, 229, 901, 208, 204, 901, 214, 901,
584: 207, 216, 202, 210, 226, 236, 206, 222, 209, 217,
585: 901, 206, 226, 227, 216, 215, 218, 231, 220, 901,
586: 214, 234, 901, 235, 222, 256, 229, 237, 258, 901,
587: 237, 233, 901, 243, 236, 901, 232, 237, 239, 247,
588: 274, 231, 251, 901, 258, 238, 254, 243, 269, 242,
589: 245, 252, 262, 261, 253, 263, 257, 901, 292, 254,
590:
591: 293, 268, 260, 262, 901, 258, 263, 265, 273, 901,
592: 273, 268, 901, 901, 281, 901, 901, 901, 264, 901,
593: 300, 284, 274, 268, 286, 272, 278, 274, 276, 278,
594: 275, 901, 281, 279, 294, 901, 291, 286, 901, 901,
595: 901, 281, 286, 298, 290, 901, 299, 294, 305, 302,
596: 901, 288, 304, 318, 306, 901, 311, 901, 308, 294,
597: 901, 308, 304, 293, 312, 306, 315, 304, 317, 317,
598: 305, 901, 306, 321, 901, 901, 308, 309, 316, 329,
599: 325, 326, 324, 901, 901, 901, 316, 332, 317, 332,
600: 329, 325, 901, 316, 321, 325, 340, 331, 901, 328,
601:
602: 341, 338, 901, 343, 336, 345, 901, 348, 339, 348,
603: 335, 901, 901, 901, 392, 394, 395, 397, 399
604: };
605:
606: static yyconst flex_int16_t yy_def[820] =
607: { 0,
608: 814, 1, 814, 814, 814, 814, 815, 814, 814, 814,
609: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
610: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
611: 814, 814, 814, 814, 814, 814, 814, 814, 815, 814,
612: 815, 814, 814, 814, 814, 814, 814, 814, 814, 814,
613: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,

```

614: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
615: 814, 814, 814, 814, 816, 814, 814, 814, 814, 814,
616: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
617: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
618:
619: 814, 814, 814, 814, 814, 815, 814, 814, 814, 814,
620: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
621: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
622: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
623: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
624: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
625: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
626: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
627: 814, 814, 817, 814, 814, 814, 814, 814, 814, 814,
628: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
629:
630: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
631: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
632: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
633: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
634: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
635: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
636: 814, 814, 814, 814, 814, 818, 817, 818, 819, 814,
637: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
638: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
639: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
640:
641: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
642: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
643: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
644: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
645: 814, 814, 814, 814, 814, 814, 814, 814, 814, 818,
646: 817, 819, 819, 817, 814, 814, 814, 814, 814, 814,
647: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
648: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
649: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
650: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
651:
652: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
653: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
654: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,

655: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
656: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
657: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
658: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
659: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
660: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
661: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
662:
663: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
664: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
665: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
666: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
667: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
668: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
669: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
670: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
671: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
672: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
673:
674: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
675: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
676: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
677: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
678: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
679: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
680: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
681: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
682: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
683: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
684:
685: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
686: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
687: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
688: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
689: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
690: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
691: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
692: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
693: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
694: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
695:

```

696:    814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
697:    814, 814, 814, 0, 814, 814, 814, 814, 814
698: };
699:
700: static yyconst flex_int16_t yy_nxt[965] =
701: { 0,
702:    4, 5, 6, 5, 4, 7, 4, 8, 9, 10,
703:    4, 11, 12, 13, 14, 4, 4, 4, 4, 4,
704:    4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
705:    15, 4, 4, 4, 4, 4, 4, 4, 16, 4,
706:    17, 18, 19, 20, 4, 4, 21, 22, 23, 24,
707:    25, 26, 27, 28, 29, 30, 31, 32, 33, 4,
708:    34, 35, 36, 38, 40, 38, 42, 43, 44, 61,
709:    45, 62, 63, 67, 38, 69, 38, 662, 37, 70,
710:    64, 37, 65, 37, 37, 71, 68, 72, 83, 74,
711:    37, 78, 66, 37, 97, 37, 37, 73, 37, 84,
712:
713:    37, 75, 37, 37, 37, 41, 37, 37, 46, 79,
714:    76, 663, 47, 80, 44, 88, 45, 37, 48, 49,
715:    50, 51, 81, 52, 40, 53, 82, 54, 85, 55,
716:    56, 89, 57, 58, 90, 37, 37, 59, 664, 98,
717:    93, 86, 94, 99, 87, 37, 37, 100, 101, 91,
718:    37, 37, 95, 92, 102, 103, 96, 37, 104, 122,
719:    106, 131, 124, 132, 60, 123, 125, 144, 105, 149,
720:    171, 665, 105, 150, 172, 223, 156, 153, 126, 145,
721:    666, 37, 108, 109, 110, 146, 111, 157, 112, 241,
722:    113, 185, 114, 115, 187, 116, 117, 186, 188, 194,
723:
724:    118, 195, 202, 242, 224, 271, 329, 267, 268, 203,
725:    189, 269, 272, 330, 340, 351, 267, 416, 351, 352,
726:    352, 341, 353, 376, 420, 351, 667, 119, 377, 269,
727:    417, 267, 267, 436, 465, 269, 269, 421, 437, 492,
728:    657, 466, 668, 669, 670, 671, 672, 493, 673, 674,
729:    675, 676, 658, 677, 678, 679, 680, 681, 682, 683,
730:    684, 685, 686, 687, 688, 689, 690, 691, 692, 693,
731:    694, 695, 696, 697, 698, 699, 700, 701, 702, 703,
732:    704, 705, 706, 707, 708, 709, 710, 711, 712, 713,
733:    714, 715, 716, 717, 718, 719, 720, 721, 722, 723,
734:
735:    724, 725, 726, 727, 728, 729, 730, 731, 732, 733,
736:    734, 735, 736, 737, 738, 739, 740, 523, 741, 742,

```

737: 743, 744, 745, 746, 747, 748, 749, 750, 751, 752,
 738: 753, 754, 755, 756, 757, 572, 758, 759, 760, 761,
 739: 762, 763, 764, 765, 766, 767, 768, 769, 770, 771,
 740: 772, 773, 774, 775, 776, 777, 778, 779, 780, 781,
 741: 782, 783, 784, 785, 786, 787, 788, 789, 790, 791,
 742: 792, 793, 794, 795, 796, 797, 798, 799, 800, 801,
 743: 802, 803, 804, 805, 806, 807, 808, 809, 810, 811,
 744: 812, 813, 39, 39, 105, 266, 266, 350, 350, 354,
 745:
 746: 354, 661, 660, 659, 656, 655, 654, 653, 652, 651,
 747: 650, 649, 648, 647, 646, 645, 644, 643, 642, 641,
 748: 640, 564, 639, 638, 637, 636, 635, 634, 633, 632,
 749: 631, 630, 629, 628, 627, 626, 625, 624, 623, 622,
 750: 621, 620, 619, 618, 617, 616, 615, 614, 613, 612,
 751: 611, 610, 609, 608, 607, 606, 605, 604, 603, 602,
 752: 601, 600, 599, 598, 597, 596, 595, 594, 593, 592,
 753: 499, 591, 590, 589, 588, 587, 586, 585, 584, 583,
 754: 582, 581, 580, 579, 578, 577, 576, 575, 574, 573,
 755: 572, 571, 570, 569, 568, 567, 566, 565, 564, 563,
 756:
 757: 562, 561, 560, 559, 558, 557, 556, 555, 554, 553,
 758: 552, 551, 550, 549, 548, 547, 546, 545, 544, 543,
 759: 542, 541, 540, 539, 538, 537, 536, 535, 534, 533,
 760: 532, 531, 530, 529, 528, 527, 526, 525, 524, 523,
 761: 522, 521, 520, 519, 518, 517, 516, 515, 514, 513,
 762: 512, 511, 510, 509, 508, 507, 506, 432, 505, 504,
 763: 503, 502, 501, 500, 499, 498, 497, 496, 495, 494,
 764: 491, 490, 489, 488, 487, 486, 485, 484, 483, 482,
 765: 481, 480, 479, 478, 477, 476, 475, 474, 473, 472,
 766: 471, 470, 469, 468, 467, 464, 463, 462, 461, 460,
 767:
 768: 459, 458, 457, 456, 455, 454, 453, 452, 451, 450,
 769: 449, 448, 447, 446, 445, 444, 443, 442, 441, 440,
 770: 439, 438, 435, 434, 433, 432, 431, 814, 430, 429,
 771: 428, 427, 426, 425, 424, 423, 422, 419, 418, 415,
 772: 414, 413, 412, 411, 410, 409, 408, 407, 406, 405,
 773: 404, 403, 402, 401, 400, 399, 398, 397, 396, 395,
 774: 394, 393, 392, 391, 390, 389, 388, 387, 386, 385,
 775: 384, 383, 382, 381, 380, 379, 378, 375, 374, 373,
 776: 372, 371, 370, 369, 368, 367, 366, 365, 364, 363,
 777: 362, 361, 360, 359, 358, 357, 356, 355, 349, 348,

```

778:
779: 347, 346, 345, 344, 343, 342, 339, 338, 337, 336,
780: 335, 334, 333, 332, 331, 328, 327, 326, 325, 324,
781: 323, 322, 321, 320, 319, 318, 317, 316, 315, 314,
782: 313, 312, 311, 310, 309, 308, 307, 306, 305, 304,
783: 303, 302, 301, 300, 299, 298, 297, 296, 295, 294,
784: 293, 292, 291, 290, 289, 288, 287, 286, 285, 284,
785: 283, 282, 281, 280, 279, 278, 277, 276, 275, 274,
786: 273, 270, 265, 264, 263, 262, 261, 260, 259, 258,
787: 257, 256, 255, 254, 253, 252, 251, 250, 249, 248,
788: 247, 246, 245, 244, 243, 240, 239, 238, 237, 236,
789:
790: 235, 234, 233, 232, 231, 230, 229, 228, 227, 226,
791: 225, 222, 221, 220, 219, 218, 217, 216, 215, 214,
792: 213, 212, 211, 210, 209, 208, 207, 206, 205, 204,
793: 201, 200, 199, 198, 197, 196, 193, 192, 191, 190,
794: 184, 183, 40, 182, 181, 180, 179, 178, 177, 176,
795: 175, 174, 173, 170, 169, 168, 167, 166, 165, 164,
796: 163, 162, 161, 160, 159, 158, 155, 154, 152, 151,
797: 148, 147, 143, 142, 141, 140, 139, 138, 137, 136,
798: 135, 134, 133, 130, 129, 128, 127, 121, 120, 107,
799: 42, 42, 40, 105, 37, 77, 37, 37, 37, 814,
800:
801: 3, 814, 814, 814, 814, 814, 814, 814, 814, 814,
802: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
803: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
804: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
805: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
806: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
807: 814, 814, 814, 814
808: };
809:
810: static yyconst flex_int16_t yy_chk[965] =
811: { 0,
812: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
813: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
814: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
815: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
816: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
817: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
818: 1, 1, 1, 5, 7, 5, 8, 9, 10, 14,

```

819: 10, 15, 16, 17, 38, 18, 38, 616, 16, 18,
 820: 16, 17, 16, 19, 20, 19, 17, 19, 25, 20,
 821: 15, 23, 16, 26, 31, 23, 25, 19, 31, 26,
 822:
 823: 9, 21, 14, 8, 28, 7, 21, 10, 11, 24,
 824: 21, 617, 11, 24, 45, 28, 45, 24, 11, 11,
 825: 11, 11, 24, 11, 41, 11, 24, 11, 27, 11,
 826: 11, 29, 11, 11, 29, 32, 27, 11, 618, 32,
 827: 30, 27, 30, 33, 27, 34, 30, 33, 34, 29,
 828: 11, 33, 30, 29, 34, 35, 30, 29, 35, 50,
 829: 41, 56, 51, 56, 11, 50, 51, 69, 77, 73,
 830: 94, 619, 70, 73, 94, 142, 80, 77, 51, 69,
 831: 621, 35, 47, 47, 47, 70, 47, 80, 47, 159,
 832: 47, 109, 47, 47, 110, 47, 47, 109, 110, 115,
 833:
 834: 47, 115, 122, 159, 142, 185, 245, 183, 183, 122,
 835: 110, 183, 185, 245, 256, 266, 267, 333, 268, 266,
 836: 267, 256, 268, 292, 337, 350, 622, 47, 292, 350,
 837: 333, 351, 354, 360, 390, 351, 354, 337, 360, 418,
 838: 612, 390, 623, 624, 626, 628, 629, 418, 630, 631,
 839: 632, 633, 612, 634, 636, 637, 639, 641, 642, 643,
 840: 644, 645, 646, 647, 648, 649, 650, 652, 653, 654,
 841: 655, 656, 657, 658, 659, 661, 662, 664, 665, 666,
 842: 667, 668, 669, 671, 672, 674, 675, 677, 678, 679,
 843: 680, 681, 682, 683, 685, 686, 687, 688, 689, 690,
 844:
 845: 691, 692, 693, 694, 695, 696, 697, 699, 700, 701,
 846: 702, 703, 704, 706, 707, 708, 709, 711, 712, 715,
 847: 719, 721, 722, 723, 724, 725, 726, 727, 728, 729,
 848: 730, 731, 733, 734, 735, 737, 738, 742, 743, 744,
 849: 745, 747, 748, 749, 750, 752, 753, 754, 755, 757,
 850: 759, 760, 762, 763, 764, 765, 766, 767, 768, 769,
 851: 770, 771, 773, 774, 777, 778, 779, 780, 781, 782,
 852: 783, 787, 788, 789, 790, 791, 792, 794, 795, 796,
 853: 797, 798, 800, 801, 802, 804, 805, 806, 808, 809,
 854: 810, 811, 815, 815, 816, 817, 817, 818, 818, 819,
 855:
 856: 819, 615, 614, 613, 611, 608, 607, 606, 605, 604,
 857: 603, 602, 601, 600, 599, 598, 597, 596, 595, 594,
 858: 593, 592, 591, 590, 589, 588, 586, 584, 583, 582,
 859: 581, 579, 578, 577, 576, 575, 574, 573, 571, 570,

860: 569, 568, 567, 566, 565, 563, 562, 561, 560, 559,
861: 556, 555, 554, 553, 552, 551, 550, 549, 548, 547,
862: 546, 545, 542, 541, 540, 539, 538, 537, 536, 534,
863: 533, 532, 530, 529, 528, 527, 526, 525, 524, 522,
864: 521, 520, 519, 518, 517, 515, 514, 513, 512, 509,
865: 508, 507, 506, 505, 504, 503, 502, 501, 500, 498,
866:
867: 497, 496, 495, 494, 493, 492, 491, 490, 489, 488,
868: 487, 486, 485, 483, 481, 480, 479, 478, 476, 475,
869: 474, 473, 472, 471, 470, 469, 468, 467, 466, 465,
870: 464, 463, 462, 461, 459, 458, 457, 456, 453, 452,
871: 451, 450, 448, 447, 446, 445, 444, 443, 442, 441,
872: 440, 439, 438, 437, 436, 434, 433, 431, 430, 429,
873: 428, 427, 426, 425, 424, 423, 422, 421, 420, 419,
874: 417, 416, 415, 414, 413, 412, 411, 410, 409, 408,
875: 407, 406, 405, 403, 401, 400, 399, 398, 397, 396,
876: 395, 394, 393, 392, 391, 388, 387, 386, 385, 384,
877:
878: 383, 382, 381, 380, 379, 378, 377, 376, 374, 373,
879: 372, 371, 370, 369, 368, 367, 366, 365, 364, 363,
880: 362, 361, 359, 358, 356, 355, 353, 352, 348, 347,
881: 346, 343, 342, 341, 340, 339, 338, 336, 335, 332,
882: 331, 330, 329, 328, 327, 326, 325, 324, 323, 322,
883: 321, 320, 319, 318, 317, 316, 315, 314, 313, 312,
884: 311, 310, 309, 306, 305, 304, 303, 302, 301, 300,
885: 299, 298, 297, 296, 295, 294, 293, 291, 290, 288,
886: 286, 285, 284, 283, 282, 281, 280, 279, 278, 277,
887: 276, 275, 274, 273, 272, 271, 270, 269, 264, 263,
888:
889: 262, 261, 260, 259, 258, 257, 255, 254, 253, 252,
890: 251, 249, 248, 247, 246, 244, 242, 241, 240, 239,
891: 238, 237, 236, 234, 233, 232, 231, 230, 229, 228,
892: 227, 226, 225, 224, 223, 221, 220, 219, 218, 217,
893: 216, 215, 214, 213, 212, 211, 210, 209, 208, 207,
894: 206, 205, 204, 203, 202, 201, 200, 199, 198, 197,
895: 196, 195, 194, 193, 192, 191, 190, 189, 188, 187,
896: 186, 184, 182, 181, 180, 179, 178, 177, 176, 175,
897: 174, 173, 172, 171, 170, 169, 168, 167, 166, 165,
898: 164, 163, 162, 161, 160, 158, 157, 156, 155, 154,
899:
900: 153, 152, 151, 150, 149, 148, 147, 146, 145, 144,

```

901: 143, 141, 140, 139, 138, 137, 136, 135, 134, 133,
902: 132, 131, 130, 129, 128, 127, 126, 125, 124, 123,
903: 121, 120, 119, 118, 117, 116, 114, 113, 112, 111,
904: 108, 107, 106, 104, 103, 102, 101, 100, 99, 98,
905: 97, 96, 95, 93, 92, 91, 90, 89, 88, 87,
906: 86, 85, 84, 83, 82, 81, 79, 78, 76, 74,
907: 72, 71, 68, 67, 66, 65, 64, 63, 62, 60,
908: 59, 58, 57, 55, 54, 53, 52, 49, 48, 46,
909: 44, 42, 39, 37, 36, 22, 13, 12, 4, 3,
910:
911: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
912: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
913: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
914: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
915: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
916: 814, 814, 814, 814, 814, 814, 814, 814, 814, 814,
917: 814, 814, 814, 814
918: };
919:
920: static yy_state_type yy_last_accepting_state;
921: static char *yy_last_accepting_cpos;
922:
923: extern int xTEDS_flex_debug;
924: int xTEDS_flex_debug = 0;
925:
926: /* The intent behind this definition is that it'll catch
927:  * any uses of REJECT which flex missed.
928:  */
929: #define REJECT reject_used_but_not_detected
930: #define yymore() yymore_used_but_not_detected
931: #define YY_MORE_ADJ 0
932: #define YY_RESTORE_YY_MORE_OFFSET
933: char *xTEDStext;
934: #line 1 "xTEDS.l"
935: #line 2 "xTEDS.l"
936: #define _GNU_SOURCE
937: #include <stdio.h>
938: #include <stdlib.h>
939: #include <string.h>
940:
941: #ifndef WIN32

```

```

942: # ifndef strndup
943: #  include "../MemoryUtils.h"
944: # endif
945: #else
946: # include "unistd.h"
947: #endif
948:
949: #include "xTEDS.tab.h"
950: int line_count = 1;
951: #line 952 "lex.xTEDS.c"
952:
953: #define INITIAL 0
954:
955: #ifndef YY_NO_UNISTD_H
956: /* Special case for "unistd.h", since it is non-ANSI. We include it way
957:  * down here because we want the user's section 1 to have been scanned first.
958:  * The user has a chance to override it with an option.
959:  */
960: #include <unistd.h>
961: #endif
962:
963: #ifndef YY_EXTRA_TYPE
964: #define YY_EXTRA_TYPE void *
965: #endif
966:
967: static int yy_init_globals (void );
968:
969: /* Macros after this point can all be overridden by user definitions in
970:  * section 1.
971:  */
972:
973: #ifndef YY_SKIP_YWRAP
974: #ifdef __cplusplus
975: extern "C" int xTEDSwrap (void );
976: #else
977: extern int xTEDSwrap (void );
978: #endif
979: #endif
980:
981: #ifndef yytext_ptr
982: static void yy_flex_strncpy (char *,yyconst char *,int );

```



```

983: #endif
984:
985: #ifdef YY_NEED_STRLEN
986: static int yy_flex_strlen (yyconst char * );
987: #endif
988:
989: #ifndef YY_NO_INPUT
990:
991: #ifdef __cplusplus
992: static int yyinput (void );
993: #else
994: static int input (void );
995: #endif
996:
997: #endif
998:
999: /* Amount of stuff to slurp up with each read. */
1000: #ifndef YY_READ_BUF_SIZE
1001: #define YY_READ_BUF_SIZE 8192
1002: #endif
1003:
1004: /* Copy whatever the last rule matched to the standard output. */
1005: #ifndef ECHO
1006: /* This used to be an fputs(), but since the string might contain NUL's,
1007:  * we now use fwrite().
1008:  */
1009: #define ECHO (void) fwrite( xTEDStext, xTEDSleng, 1, xTEDSout )
1010: #endif
1011:
1012: /* Gets input and stuffs it into "buf".  number of characters read, or YY_NULL,
1013:  * is returned in "result".
1014:  */
1015: #ifndef YY_INPUT
1016: #define YY_INPUT(buf,result,max_size) \
1017:   if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
1018:     { \
1019:       int c = '*'; \
1020:       size_t n; \
1021:       for ( n = 0; n < max_size && \
1022:             (c = getc( xTEDSin )) != EOF && c != '\n'; ++n ) \
1023:         buf[n] = (char) c; \

```

```

1024:     if ( c == '\n' ) \
1025:         buf[n++] = (char) c; \
1026:     if ( c == EOF && ferror( xTEDSin ) ) \
1027:         YY_FATAL_ERROR( "input in flex scanner failed" ); \
1028:     result = n; \
1029: } \
1030: else \
1031: { \
1032:     errno=0; \
1033:     while ( (result = fread(buf, 1, max_size, xTEDSin))==0 && ferror(xTEDSin)) \
1034:         { \
1035:             if( errno != EINTR) \
1036:                 { \
1037:                     YY_FATAL_ERROR( "input in flex scanner failed" ); \
1038:                     break; \
1039:                 } \
1040:             errno=0; \
1041:             clearerr(xTEDSin); \
1042:         } \
1043:     } \
1044: \
1045:
1046: #endif
1047:
1048: /* No semi-colon after return; correct usage is to write "yyterminate();" -
1049:  * we don't want an extra ';' after the "return" because that will cause
1050:  * some compilers to complain about unreachable statements.
1051:  */
1052: #ifndef yyterminate
1053: #define yyterminate() return YY_NULL
1054: #endif
1055:
1056: /* Number of entries by which start-condition stack grows. */
1057: #ifndef YY_START_STACK_INCR
1058: #define YY_START_STACK_INCR 25
1059: #endif
1060:
1061: /* Report a fatal error. */
1062: #ifndef YY_FATAL_ERROR
1063: #define YY_FATAL_ERROR(msg) yy_fatal_error( msg )
1064: #endif

```

```

1065:
1066: /* end tables serialization structures and prototypes */
1067:
1068: /* Default declaration of generated scanner - a define so the user can
1069:  * easily add parameters.
1070:  */
1071: #ifndef YY_DECL
1072: #define YY_DECL_IS_OURS 1
1073:
1074: extern int xTEDSlex (void);
1075:
1076: #define YY_DECL int xTEDSlex (void)
1077: #endif /* !YY_DECL */
1078:
1079: /* Code executed at the beginning of each rule, after xTEDStext and xTEDSleng
1080:  * have been set up.
1081:  */
1082: #ifndef YY_USER_ACTION
1083: #define YY_USER_ACTION
1084: #endif
1085:
1086: /* Code executed at the end of each rule. */
1087: #ifndef YY_BREAK
1088: #define YY_BREAK break;
1089: #endif
1090:
1091: #define YY_RULE_SETUP \
1092:   YY_USER_ACTION
1093:
1094: /** The main scanner function which does all the work.
1095:  */
1096: YY_DECL
1097: {
1098:   register yy_state_type yy_current_state;
1099:   register char *yy_cp, *yy_bp;
1100:   register int yy_act;
1101:
1102: #line 21 "xTEDS.l"
1103:
1104:
1105: #line 1106 "lex.xTEDS.c"

```

```

1106:
1107:  if ( !(yy_init) )
1108:      {
1109:          (yy_init) = 1;
1110:
1111:  #ifdef YY_USER_INIT
1112:      YY_USER_INIT;
1113:  #endif
1114:
1115:      if ( ! (yy_start) )
1116:          (yy_start) = 1; /* first start state */
1117:
1118:      if ( ! xTEDSin )
1119:          xTEDSin = stdin;
1120:
1121:      if ( ! xTEDSout )
1122:          xTEDSout = stdout;
1123:
1124:      if ( ! YY_CURRENT_BUFFER ) {
1125:          xTEDSensure_buffer_stack ();
1126:          YY_CURRENT_BUFFER_LVALUE =
1127:              xTEDS_create_buffer(xTEDSin,YY_BUF_SIZE );
1128:      }
1129:
1130:      xTEDS_load_buffer_state( );
1131:  }
1132:
1133:  while ( 1 )      /* loops until end-of-file is reached */
1134:      {
1135:          yy_cp = (yy_c_buf_p);
1136:
1137:          /* Support of xTEDStext. */
1138:          *yy_cp = (yy_hold_char);
1139:
1140:          /* yy_bp points to the position in yy_ch_buf of the start of
1141:           * the current run.
1142:           */
1143:          yy_bp = yy_cp;
1144:
1145:          yy_current_state = (yy_start);
1146:  yy_match:

```

```

1147:    do
1148:        {
1149:            register YY_CHAR yy_c = yy_ec[YY_SC_TO_UI(*yy_cp)];
1150:            if ( yy_accept[yy_current_state] )
1151:                {
1152:                    (yy_last_accepting_state) = yy_current_state;
1153:                    (yy_last_accepting_cpos) = yy_cp;
1154:                }
1155:            while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
1156:                {
1157:                    yy_current_state = (int) yy_def[yy_current_state];
1158:                    if ( yy_current_state >= 815 )
1159:                        yy_c = yy_meta[(unsigned int) yy_c];
1160:                }
1161:            yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
1162:            ++yy_cp;
1163:        }
1164:    while ( yy_current_state != 814 );
1165:    yy_cp = (yy_last_accepting_cpos);
1166:    yy_current_state = (yy_last_accepting_state);
1167:
1168: yy_find_action:
1169:    yy_act = yy_accept[yy_current_state];
1170:
1171:    YY_DO_BEFORE_ACTION;
1172:
1173: do_action:    /* This label is used only to access EOF actions. */
1174:
1175:    switch ( yy_act )
1176:    { /* beginning of action switch */
1177:        case 0: /* must back up */
1178:            /* undo the effects of YY_DO_BEFORE_ACTION */
1179:            *yy_cp = (yy_hold_char);
1180:            yy_cp = (yy_last_accepting_cpos);
1181:            yy_current_state = (yy_last_accepting_state);
1182:            goto yy_find_action;
1183:
1184: case 1:
1185: YY_RULE_SETUP
1186: #line 23 "xTEDS.l"
1187: {return EQUAL_SY;}

```

1188: YY_BREAK
1189: case 2:
1190: YY_RULE_SETUP
1191: #line 24 "xTEDS.l"
1192: {return CLOSE_SY;}
1193: YY_BREAK
1194: case 3:
1195: YY_RULE_SETUP
1196: #line 25 "xTEDS.l"
1197: {return SLASHCLOSE_SY;}
1198: YY_BREAK
1199: case 4:
1200: YY_RULE_SETUP
1201: #line 26 "xTEDS.l"
1202: {return CLOSE_XML_SY;}
1203: YY_BREAK
1204: case 5:
1205: YY_RULE_SETUP
1206: #line 28 "xTEDS.l"
1207: {return OPEN_XML_SY;}
1208: YY_BREAK
1209: case 6:
1210: YY_RULE_SETUP
1211: #line 29 "xTEDS.l"
1212: {return CLOSE_xTEDS_SY;}
1213: YY_BREAK
1214: case 7:
1215: YY_RULE_SETUP
1216: #line 30 "xTEDS.l"
1217: {return OPEN_xTEDS_SY;}
1218: YY_BREAK
1219: case 8:
1220: YY_RULE_SETUP
1221: #line 31 "xTEDS.l"
1222: {return OPEN_APP_SY;}
1223: YY_BREAK
1224: case 9:
1225: YY_RULE_SETUP
1226: #line 32 "xTEDS.l"
1227: {return CLOSE_APP_SY;}
1228: YY_BREAK

1229: case 10:
1230: YY_RULE_SETUP
1231: #line 33 "xTEDS.l"
1232: {return OPEN_VAR_SY;}
1233: YY_BREAK
1234: case 11:
1235: YY_RULE_SETUP
1236: #line 34 "xTEDS.l"
1237: {return CLOSE_VAR_SY;}
1238: YY_BREAK
1239: case 12:
1240: YY_RULE_SETUP
1241: #line 35 "xTEDS.l"
1242: {return OPEN_DRANGE_SY;}
1243: YY_BREAK
1244: case 13:
1245: YY_RULE_SETUP
1246: #line 36 "xTEDS.l"
1247: {return CLOSE_DRANGE_SY;}
1248: YY_BREAK
1249: case 14:
1250: YY_RULE_SETUP
1251: #line 37 "xTEDS.l"
1252: {return OPEN_OPTION_SY;}
1253: YY_BREAK
1254: case 15:
1255: YY_RULE_SETUP
1256: #line 38 "xTEDS.l"
1257: {return CLOSE_OPTION_SY;}
1258: YY_BREAK
1259: case 16:
1260: YY_RULE_SETUP
1261: #line 39 "xTEDS.l"
1262: {return OPEN_CURVE_SY;}
1263: YY_BREAK
1264: case 17:
1265: YY_RULE_SETUP
1266: #line 40 "xTEDS.l"
1267: {return CLOSE_CURVE_SY;}
1268: YY_BREAK
1269: case 18:

1270: YY_RULE_SETUP
1271: #line 41 "xTEDS.l"
1272: {return OPEN_COEFF_SY;}
1273: YY_BREAK
1274: case 19:
1275: YY_RULE_SETUP
1276: #line 42 "xTEDS.l"
1277: {return CLOSE_COEFF_SY;}
1278: YY_BREAK
1279: case 20:
1280: YY_RULE_SETUP
1281: #line 43 "xTEDS.l"
1282: {return OPEN_DATA_MSG_SY;}
1283: YY_BREAK
1284: case 21:
1285: YY_RULE_SETUP
1286: #line 44 "xTEDS.l"
1287: {return CLOSE_DATA_MSG_SY;}
1288: YY_BREAK
1289: case 22:
1290: YY_RULE_SETUP
1291: #line 45 "xTEDS.l"
1292: {return OPEN_VARIABLE_REF_SY;}
1293: YY_BREAK
1294: case 23:
1295: YY_RULE_SETUP
1296: #line 46 "xTEDS.l"
1297: {return CLOSE_VARIABLE_REF_SY;}
1298: YY_BREAK
1299: case 24:
1300: YY_RULE_SETUP
1301: #line 47 "xTEDS.l"
1302: {return OPEN_COMMAND_MSG_SY;}
1303: YY_BREAK
1304: case 25:
1305: YY_RULE_SETUP
1306: #line 48 "xTEDS.l"
1307: {return CLOSE_COMMAND_MSG_SY;}
1308: YY_BREAK
1309: case 26:
1310: YY_RULE_SETUP

1311: #line 49 "xTEDS.I"
1312: {return OPEN_DEVICE_SY;}
1313: YY_BREAK
1314: case 27:
1315: YY_RULE_SETUP
1316: #line 50 "xTEDS.I"
1317: {return CLOSE_DEVICE_SY;}
1318: YY_BREAK
1319: case 28:
1320: YY_RULE_SETUP
1321: #line 51 "xTEDS.I"
1322: {return OPEN_INTERFACE_SY;}
1323: YY_BREAK
1324: case 29:
1325: YY_RULE_SETUP
1326: #line 52 "xTEDS.I"
1327: {return CLOSE_INTERFACE_SY;}
1328: YY_BREAK
1329: case 30:
1330: YY_RULE_SETUP
1331: #line 53 "xTEDS.I"
1332: {return OPEN_COMMAND_SY;}
1333: YY_BREAK
1334: case 31:
1335: YY_RULE_SETUP
1336: #line 54 "xTEDS.I"
1337: {return CLOSE_COMMAND_SY;}
1338: YY_BREAK
1339: case 32:
1340: YY_RULE_SETUP
1341: #line 55 "xTEDS.I"
1342: {return OPEN_NOTIFICATION_SY;}
1343: YY_BREAK
1344: case 33:
1345: YY_RULE_SETUP
1346: #line 56 "xTEDS.I"
1347: {return CLOSE_NOTIFICATION_SY;}
1348: YY_BREAK
1349: case 34:
1350: YY_RULE_SETUP
1351: #line 57 "xTEDS.I"

1352: {return OPEN_REQUEST_SY;}
1353: YY_BREAK
1354: case 35:
1355: YY_RULE_SETUP
1356: #line 58 "xTEDS.l"
1357: {return CLOSE_REQUEST_SY;}
1358: YY_BREAK
1359: case 36:
1360: YY_RULE_SETUP
1361: #line 59 "xTEDS.l"
1362: {return OPEN_FAULT_MSG_SY;}
1363: YY_BREAK
1364: case 37:
1365: YY_RULE_SETUP
1366: #line 60 "xTEDS.l"
1367: {return CLOSE_FAULT_MSG_SY;}
1368: YY_BREAK
1369: case 38:
1370: YY_RULE_SETUP
1371: #line 61 "xTEDS.l"
1372: {return OPEN_QUALIFIER_SY;}
1373: YY_BREAK
1374: case 39:
1375: YY_RULE_SETUP
1376: #line 62 "xTEDS.l"
1377: {return CLOSE_QUALIFIER_SY;}
1378: YY_BREAK
1379: case 40:
1380: YY_RULE_SETUP
1381: #line 63 "xTEDS.l"
1382: {return OPEN_LOCATION_SY;}
1383: YY_BREAK
1384: case 41:
1385: YY_RULE_SETUP
1386: #line 64 "xTEDS.l"
1387: {return CLOSE_LOCATION_SY;}
1388: YY_BREAK
1389: case 42:
1390: YY_RULE_SETUP
1391: #line 65 "xTEDS.l"
1392: {return OPEN_ORIENTATION_SY;}

```

1393: YY_BREAK
1394: case 43:
1395: YY_RULE_SETUP
1396: #line 66 "xTEDS.l"
1397: {return CLOSE_ORIENTATION_SY;}
1398: YY_BREAK
1399: case 44:
1400: YY_RULE_SETUP
1401: #line 68 "xTEDS.l"
1402: {return INVALID_VALUE_SY;
1403: /* Because "id" is a substring of "invalidValue", the following pattern will
1404: catch a misspelling of "invalidValue" without matching "id". This prevents the
1405: parser from incorrectly reducing grammar rules that were intended for "invalidValue"
1406: but instead "id" was matched -- specifically with <Variable> attributes.*/}
1407: YY_BREAK
1408: case 45:
1409: YY_RULE_SETUP
1410: #line 73 "xTEDS.l"
1411: {return BAD_TERMINAL_SY;}
1412: YY_BREAK
1413: case 46:
1414: YY_RULE_SETUP
1415: #line 74 "xTEDS.l"
1416: {return NAME_SY;}
1417: YY_BREAK
1418: case 47:
1419: YY_RULE_SETUP
1420: #line 75 "xTEDS.l"
1421: {return KIND_SY;}
1422: YY_BREAK
1423: case 48:
1424: YY_RULE_SETUP
1425: #line 76 "xTEDS.l"
1426: {return ID_SY;}
1427: YY_BREAK
1428: case 49:
1429: YY_RULE_SETUP
1430: #line 77 "xTEDS.l"
1431: {return QUALIFIER_SY;}
1432: YY_BREAK
1433: case 50:

```

1434: YY_RULE_SETUP
1435: #line 78 "xTEDS.l"
1436: {return DESCRIPTION_SY;}
1437: YY_BREAK
1438: case 51:
1439: YY_RULE_SETUP
1440: #line 79 "xTEDS.l"
1441: {return MANUFACTURER_ID_SY;}
1442: YY_BREAK
1443: case 52:
1444: YY_RULE_SETUP
1445: #line 80 "xTEDS.l"
1446: {return VERSION_SY;}
1447: YY_BREAK
1448: case 53:
1449: YY_RULE_SETUP
1450: #line 81 "xTEDS.l"
1451: {return MODEL_ID_SY;}
1452: YY_BREAK
1453: case 54:
1454: YY_RULE_SETUP
1455: #line 82 "xTEDS.l"
1456: {return VERSION_LETTER_SY;}
1457: YY_BREAK
1458: case 55:
1459: YY_RULE_SETUP
1460: #line 83 "xTEDS.l"
1461: {return SERIAL_NUMBER_SY;}
1462: YY_BREAK
1463: case 56:
1464: YY_RULE_SETUP
1465: #line 84 "xTEDS.l"
1466: {return CALIBRATION_DATE_SY;}
1467: YY_BREAK
1468: case 57:
1469: YY_RULE_SETUP
1470: #line 85 "xTEDS.l"
1471: {return SENSITIVITY_AT_REF_SY;}
1472: YY_BREAK
1473: case 58:
1474: YY_RULE_SETUP

1475: #line 86 "xTEDS.I"
1476: {return REF_FREQ_SY;}
1477: YY_BREAK
1478: case 59:
1479: YY_RULE_SETUP
1480: #line 87 "xTEDS.I"
1481: {return REF_TEMP_SY;}
1482: YY_BREAK
1483: case 60:
1484: YY_RULE_SETUP
1485: #line 88 "xTEDS.I"
1486: {return MEASUREMENT_RANGE_SY;}
1487: YY_BREAK
1488: case 61:
1489: YY_RULE_SETUP
1490: #line 89 "xTEDS.I"
1491: {return ELECTRICAL_OUTPUT_SY;}
1492: YY_BREAK
1493: case 62:
1494: YY_RULE_SETUP
1495: #line 90 "xTEDS.I"
1496: {return QUALITY_FACTOR_SY;}
1497: YY_BREAK
1498: case 63:
1499: YY_RULE_SETUP
1500: #line 91 "xTEDS.I"
1501: {return TEMP_COEFF_SY;}
1502: YY_BREAK
1503: case 64:
1504: YY_RULE_SETUP
1505: #line 92 "xTEDS.I"
1506: {return DIRECTION_XYZ_SY;}
1507: YY_BREAK
1508: case 65:
1509: YY_RULE_SETUP
1510: #line 93 "xTEDS.I"
1511: {return CAL_DUE_DATE_SY;}
1512: YY_BREAK
1513: case 66:
1514: YY_RULE_SETUP
1515: #line 94 "xTEDS.I"

1516: {return POWER_REQS_SY;}
1517: YY_BREAK
1518: case 67:
1519: YY_RULE_SETUP
1520: #line 95 "xTEDS.l"
1521: {return VALUE_SY;}
1522: YY_BREAK
1523: case 68:
1524: YY_RULE_SETUP
1525: #line 96 "xTEDS.l"
1526: {return ALARM_SY;}
1527: YY_BREAK
1528: case 69:
1529: YY_RULE_SETUP
1530: #line 97 "xTEDS.l"
1531: {return MSG_ARRIVAL_SY;}
1532: YY_BREAK
1533: case 70:
1534: YY_RULE_SETUP
1535: #line 98 "xTEDS.l"
1536: {return MSG_RATE_SY;}
1537: YY_BREAK
1538: case 71:
1539: YY_RULE_SETUP
1540: #line 99 "xTEDS.l"
1541: {return PRECISION_SY;}
1542: YY_BREAK
1543: case 72:
1544: YY_RULE_SETUP
1545: #line 100 "xTEDS.l"
1546: {return RANGE_MAX_SY;}
1547: YY_BREAK
1548: case 73:
1549: YY_RULE_SETUP
1550: #line 101 "xTEDS.l"
1551: {return FORMAT_SY;}
1552: YY_BREAK
1553: case 74:
1554: YY_RULE_SETUP
1555: #line 102 "xTEDS.l"
1556: {return ACCURACY_SY;}

1557: YY_BREAK
1558: case 75:
1559: YY_RULE_SETUP
1560: #line 103 "xTEDS.l"
1561: {return RANGE_MIN_SY;}
1562: YY_BREAK
1563: case 76:
1564: YY_RULE_SETUP
1565: #line 104 "xTEDS.l"
1566: {return SCALE_FACTOR_SY;}
1567: YY_BREAK
1568: case 77:
1569: YY_RULE_SETUP
1570: #line 105 "xTEDS.l"
1571: {return UNITS_SY;}
1572: YY_BREAK
1573: case 78:
1574: YY_RULE_SETUP
1575: #line 106 "xTEDS.l"
1576: {return DEFAULT_VALUE_SY;}
1577: YY_BREAK
1578: case 79:
1579: YY_RULE_SETUP
1580: #line 107 "xTEDS.l"
1581: {return SCALE_UNITS_SY;}
1582: YY_BREAK
1583: case 80:
1584: YY_RULE_SETUP
1585: #line 108 "xTEDS.l"
1586: {return LENGTH_SY;}
1587: YY_BREAK
1588: case 81:
1589: YY_RULE_SETUP
1590: #line 109 "xTEDS.l"
1591: {return EXPONENT_SY;}
1592: YY_BREAK
1593: case 82:
1594: YY_RULE_SETUP
1595: #line 110 "xTEDS.l"
1596: {return COMPONENT_KEY_SY;}
1597: YY_BREAK

1598: case 83:
1599: YY_RULE_SETUP
1600: #line 111 "xTEDS.l"
1601: {return SPA_U_HUB_SY;}
1602: YY_BREAK
1603: case 84:
1604: YY_RULE_SETUP
1605: #line 112 "xTEDS.l"
1606: {return SPA_U_PORT_SY;}
1607: YY_BREAK
1608: case 85:
1609: YY_RULE_SETUP
1610: #line 113 "xTEDS.l"
1611: {return EXTENDS_SY;}
1612: YY_BREAK
1613: case 86:
1614: YY_RULE_SETUP
1615: #line 114 "xTEDS.l"
1616: {return MEMORY_MINIMUM_SY;}
1617: YY_BREAK
1618: case 87:
1619: YY_RULE_SETUP
1620: #line 115 "xTEDS.l"
1621: {return OPERATING_SYSTEM_SY;}
1622: YY_BREAK
1623: case 88:
1624: YY_RULE_SETUP
1625: #line 116 "xTEDS.l"
1626: {return PATH_FOR_ASSEMBLY_SY;}
1627: YY_BREAK
1628: case 89:
1629: YY_RULE_SETUP
1630: #line 117 "xTEDS.l"
1631: {return PATH_ON_SPACECRAFT_SY;}
1632: YY_BREAK
1633: case 90:
1634: YY_RULE_SETUP
1635: #line 118 "xTEDS.l"
1636: {return X_SY;}
1637: YY_BREAK
1638: case 91:

1639: YY_RULE_SETUP
1640: #line 119 "xTEDS.l"
1641: {return Y_SY;}
1642: YY_BREAK
1643: case 92:
1644: YY_RULE_SETUP
1645: #line 120 "xTEDS.l"
1646: {return Z_SY;}
1647: YY_BREAK
1648: case 93:
1649: YY_RULE_SETUP
1650: #line 121 "xTEDS.l"
1651: {return AXIS_SY;}
1652: YY_BREAK
1653: case 94:
1654: YY_RULE_SETUP
1655: #line 122 "xTEDS.l"
1656: {return ANGLE_SY;}
1657: YY_BREAK
1658: case 95:
1659: YY_RULE_SETUP
1660: #line 123 "xTEDS.l"
1661: {return ENCODING_SY;}
1662: YY_BREAK
1663: case 96:
1664: YY_RULE_SETUP
1665: #line 124 "xTEDS.l"
1666: {return STANDALONE_SY;}
1667: YY_BREAK
1668: case 97:
1669: YY_RULE_SETUP
1670: #line 125 "xTEDS.l"
1671: {return R_LOW_SY;}
1672: YY_BREAK
1673: case 98:
1674: YY_RULE_SETUP
1675: #line 126 "xTEDS.l"
1676: {return R_HIGH_SY;}
1677: YY_BREAK
1678: case 99:
1679: YY_RULE_SETUP

```

1680: #line 127 "xTEDS.l"
1681: {return Y_LOW_SY;}
1682: YY_BREAK
1683: case 100:
1684: YY_RULE_SETUP
1685: #line 128 "xTEDS.l"
1686: {return Y_HIGH_SY;}
1687: YY_BREAK
1688: case 101:
1689: YY_RULE_SETUP
1690: #line 130 "xTEDS.l"
1691: {return XMLNS_SY;}
1692: YY_BREAK
1693: case 102:
1694: YY_RULE_SETUP
1695: #line 131 "xTEDS.l"
1696: {return XMLNS_XSI_SY;}
1697: YY_BREAK
1698: case 103:
1699: YY_RULE_SETUP
1700: #line 132 "xTEDS.l"
1701: {return SCHEMA_LOCATION_SY;}
1702: YY_BREAK
1703: case 104:
1704: /* rule 104 can match eol */
1705: YY_RULE_SETUP
1706: #line 135 "xTEDS.l"
1707: {xTEDSlval.str = strdup(xTEDStext+1,strlen(xTEDStext)-2);return STRING;}
1708: YY_BREAK
1709: case 105:
1710: YY_RULE_SETUP
1711: #line 136 "xTEDS.l"
1712: {xTEDSlval.real = atof(xTEDStext); return FLOAT;}
1713: YY_BREAK
1714: case 106:
1715: YY_RULE_SETUP
1716: #line 137 "xTEDS.l"
1717: {xTEDSlval.integer = atoi(xTEDStext); return INT;}
1718: YY_BREAK
1719: case 107:
1720: /* rule 107 can match eol */

```

```

1721: YY_RULE_SETUP
1722: #line 139 "xTEDS.l"
1723: { /* ignore xteds comments */ }
1724: YY_BREAK
1725: case 108:
1726: YY_RULE_SETUP
1727: #line 141 "xTEDS.l"
1728: { /* ignore whitespace */ }
1729: YY_BREAK
1730: case 109:
1731: /* rule 109 can match eol */
1732: YY_RULE_SETUP
1733: #line 143 "xTEDS.l"
1734: { line_count++; }
1735: YY_BREAK
1736: case 110:
1737: YY_RULE_SETUP
1738: #line 145 "xTEDS.l"
1739: { /* catch all, don't print invalid chars */ }
1740: YY_BREAK
1741: case 111:
1742: YY_RULE_SETUP
1743: #line 147 "xTEDS.l"
1744: ECHO;
1745: YY_BREAK
1746: #line 1747 "lex.xTEDS.c"
1747: case YY_STATE_EOF(INITIAL):
1748: yyterminate();
1749:
1750: case YY_END_OF_BUFFER:
1751: {
1752:     /* Amount of text matched not including the EOB char. */
1753:     int yy_amount_of_matched_text = (int) (yy_cp - (yytext_ptr)) - 1;
1754:
1755:     /* Undo the effects of YY_DO_BEFORE_ACTION. */
1756:     *yy_cp = (yy_hold_char);
1757:     YY_RESTORE_YY_MORE_OFFSET
1758:
1759:     if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_NEW )
1760:     {
1761:         /* We're scanning a new file or input source. It's

```

```

1762:      * possible that this happened because the user
1763:      * just pointed xTEDSin at a new source and called
1764:      * xTEDSlex(). If so, then we have to assure
1765:      * consistency between YY_CURRENT_BUFFER and our
1766:      * globals. Here is the right place to do so, because
1767:      * this is the first action (other than possibly a
1768:      * back-up) that will match for the new input source.
1769:      */
1770:      (yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
1771:      YY_CURRENT_BUFFER_LVALUE->yy_input_file = xTEDSin;
1772:      YY_CURRENT_BUFFER_LVALUE->yy_buffer_status = YY_BUFFER_NORMAL;
1773:  }
1774:
1775:  /* Note that here we test for yy_c_buf_p "<=" to the position
1776:   * of the first EOB in the buffer, since yy_c_buf_p will
1777:   * already have been incremented past the NUL character
1778:   * (since all states make transitions on EOB to the
1779:   * end-of-buffer state). Contrast this with the test
1780:   * in input().
1781:   */
1782:  if ( (yy_c_buf_p) <= &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] )
1783:      { /* This was really a NUL. */
1784:          yy_state_type yy_next_state;
1785:
1786:          (yy_c_buf_p) = (yytext_ptr) + yy_amount_of_matched_text;
1787:
1788:          yy_current_state = yy_get_previous_state( );
1789:
1790:          /* Okay, we're now positioned to make the NUL
1791:           * transition. We couldn't have
1792:           * yy_get_previous_state() go ahead and do it
1793:           * for us because it doesn't know how to deal
1794:           * with the possibility of jamming (and we don't
1795:           * want to build jamming into it because then it
1796:           * will run more slowly).
1797:           */
1798:
1799:          yy_next_state = yy_try_NUL_trans( yy_current_state );
1800:
1801:          yy_bp = (yytext_ptr) + YY_MORE_ADJ;
1802:

```

```

1803:         if ( yy_next_state )
1804:             {
1805:                 /* Consume the NUL. */
1806:                 yy_cp = ++(yy_c_buf_p);
1807:                 yy_current_state = yy_next_state;
1808:                 goto yy_match;
1809:             }
1810:
1811:         else
1812:             {
1813:                 yy_cp = (yy_last_accepting_cpos);
1814:                 yy_current_state = (yy_last_accepting_state);
1815:                 goto yy_find_action;
1816:             }
1817:     }
1818:
1819:     else switch ( yy_get_next_buffer( ) )
1820:     {
1821:         case EOB_ACT_END_OF_FILE:
1822:             {
1823:                 (yy_did_buffer_switch_on_eof) = 0;
1824:
1825:                 if ( xTEDSwrap( ) )
1826:                     {
1827:                         /* Note: because we've taken care in
1828:                          * yy_get_next_buffer() to have set up
1829:                          * xTEDStext, we can now set up
1830:                          * yy_c_buf_p so that if some total
1831:                          * hoser (like flex itself) wants to
1832:                          * call the scanner after we return the
1833:                          * YY_NULL, it'll still work - another
1834:                          * YY_NULL will get returned.
1835:                          */
1836:                         (yy_c_buf_p) = (yytext_ptr) + YY_MORE_ADJ;
1837:
1838:                         yy_act = YY_STATE_EOF(YY_START);
1839:                         goto do_action;
1840:                     }
1841:
1842:                 else
1843:                     {

```

```

1844:             if ( ! (yy_did_buffer_switch_on_eof) )
1845:                 YY_NEW_FILE;
1846:             }
1847:             break;
1848:         }
1849:
1850:     case EOB_ACT_CONTINUE_SCAN:
1851:         (yy_c_buf_p) =
1852:             (yytext_ptr) + yy_amount_of_matched_text;
1853:
1854:         yy_current_state = yy_get_previous_state( );
1855:
1856:         yy_cp = (yy_c_buf_p);
1857:         yy_bp = (yytext_ptr) + YY_MORE_ADJ;
1858:         goto yy_match;
1859:
1860:     case EOB_ACT_LAST_MATCH:
1861:         (yy_c_buf_p) =
1862:             &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)];
1863:
1864:         yy_current_state = yy_get_previous_state( );
1865:
1866:         yy_cp = (yy_c_buf_p);
1867:         yy_bp = (yytext_ptr) + YY_MORE_ADJ;
1868:         goto yy_find_action;
1869:     }
1870:     break;
1871: }
1872:
1873: default:
1874:     YY_FATAL_ERROR(
1875:         "fatal flex scanner internal error--no action found" );
1876: } /* end of action switch */
1877: } /* end of scanning one token */
1878: } /* end of xTEDSlex */
1879:
1880: /* yy_get_next_buffer - try to read in a new buffer
1881:  *
1882:  * Returns a code representing an action:
1883:  *   EOB_ACT_LAST_MATCH -
1884:  *   EOB_ACT_CONTINUE_SCAN - continue scanning from current position

```

```

1885: *   EOB_ACT_END_OF_FILE - end of file
1886: */
1887: static int yy_get_next_buffer (void)
1888: {
1889:     register char *dest = YY_CURRENT_BUFFER_LVALUE->yy_ch_buf;
1890:     register char *source = (yytext_ptr);
1891:     register int number_to_move, i;
1892:     int ret_val;
1893:
1894:     if ( (yy_c_buf_p) > &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] )
1895:         YY_FATAL_ERROR(
1896:             "fatal flex scanner internal error--end of buffer missed" );
1897:
1898:     if ( YY_CURRENT_BUFFER_LVALUE->yy_fill_buffer == 0 )
1899:         { /* Don't try to fill the buffer, so this is an EOF. */
1900:             if ( (yy_c_buf_p) - (yytext_ptr) - YY_MORE_ADJ == 1 )
1901:                 {
1902:                     /* We matched a single character, the EOB, so
1903:                      * treat this as a final EOF.
1904:                      */
1905:                     return EOB_ACT_END_OF_FILE;
1906:                 }
1907:
1908:             else
1909:                 {
1910:                     /* We matched some text prior to the EOB, first
1911:                      * process it.
1912:                      */
1913:                     return EOB_ACT_LAST_MATCH;
1914:                 }
1915:         }
1916:
1917:     /* Try to read more data. */
1918:
1919:     /* First move last chars to start of buffer. */
1920:     number_to_move = (int) ((yy_c_buf_p) - (yytext_ptr)) - 1;
1921:
1922:     for ( i = 0; i < number_to_move; ++i )
1923:         *(dest++) = *(source++);
1924:

```

```

1925: if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_EOF_PENDING
)
1926:     /* don't do the read, it's not guaranteed to return an EOF,
1927:      * just force an EOF
1928:      */
1929:     YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars) = 0;
1930:
1931: else
1932:     {
1933:         int num_to_read =
1934:             YY_CURRENT_BUFFER_LVALUE->yy_buf_size - number_to_move - 1;
1935:
1936:         while ( num_to_read <= 0 )
1937:             { /* Not enough room in the buffer - grow it. */
1938:
1939:                 /* just a shorter name for the current buffer */
1940:                 YY_BUFFER_STATE b = YY_CURRENT_BUFFER;
1941:
1942:                 int yy_c_buf_p_offset =
1943:                     (int) ((yy_c_buf_p) - b->yy_ch_buf);
1944:
1945:                 if ( b->yy_is_our_buffer )
1946:                     {
1947:                         int new_size = b->yy_buf_size * 2;
1948:
1949:                         if ( new_size <= 0 )
1950:                             b->yy_buf_size += b->yy_buf_size / 8;
1951:                         else
1952:                             b->yy_buf_size *= 2;
1953:
1954:                         b->yy_ch_buf = (char *)
1955:                             /* Include room in for 2 EOB chars. */
1956:                             xTEDSrealloc((void *) b->yy_ch_buf,b->yy_buf_size + 2 );
1957:                     }
1958:                 else
1959:                     /* Can't grow it, we don't own it. */
1960:                     b->yy_ch_buf = 0;
1961:
1962:                 if ( ! b->yy_ch_buf )
1963:                     YY_FATAL_ERROR(
1964:                         "fatal error - scanner input buffer overflow" );

```



```

1965:
1966:     (yy_c_buf_p) = &b->yy_ch_buf[yy_c_buf_p_offset];
1967:
1968:     num_to_read = YY_CURRENT_BUFFER_LVALUE->yy_buf_size -
1969:         number_to_move - 1;
1970:
1971:     }
1972:
1973:     if ( num_to_read > YY_READ_BUF_SIZE )
1974:         num_to_read = YY_READ_BUF_SIZE;
1975:
1976:     /* Read in more data. */
1977:     YY_INPUT( (&YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[number_to_move]),
1978:         (yy_n_chars), (size_t) num_to_read );
1979:
1980:     YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
1981:     }
1982:
1983:     if ( (yy_n_chars) == 0 )
1984:     {
1985:         if ( number_to_move == YY_MORE_ADJ )
1986:         {
1987:             ret_val = EOB_ACT_END_OF_FILE;
1988:             xTEDSrestart(xTEDSin );
1989:         }
1990:
1991:         else
1992:         {
1993:             ret_val = EOB_ACT_LAST_MATCH;
1994:             YY_CURRENT_BUFFER_LVALUE->yy_buffer_status =
1995:                 YY_BUFFER_EOF_PENDING;
1996:         }
1997:     }
1998:
1999:     else
2000:         ret_val = EOB_ACT_CONTINUE_SCAN;
2001:
2002:     (yy_n_chars) += number_to_move;
2003:     YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] =
YY_END_OF_BUFFER_CHAR;
=

```

```

2004: YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)      +      1]      =
YY_END_OF_BUFFER_CHAR;
2005:
2006: (yytext_ptr) = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[0];
2007:
2008: return ret_val;
2009: }
2010:
2011: /* yy_get_previous_state - get the state just before the EOB char was reached */
2012:
2013: static yy_state_type yy_get_previous_state (void)
2014: {
2015: register yy_state_type yy_current_state;
2016: register char *yy_cp;
2017:
2018: yy_current_state = (yy_start);
2019:
2020: for ( yy_cp = (yytext_ptr) + YY_MORE_ADJ; yy_cp < (yy_c_buf_p); ++yy_cp )
2021: {
2022: register YY_CHAR yy_c = (*yy_cp ? yy_ec[YY_SC_TO_UI(*yy_cp)] : 1);
2023: if ( yy_accept[yy_current_state] )
2024: {
2025: (yy_last_accepting_state) = yy_current_state;
2026: (yy_last_accepting_cpos) = yy_cp;
2027: }
2028: while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
2029: {
2030: yy_current_state = (int) yy_def[yy_current_state];
2031: if ( yy_current_state >= 815 )
2032: yy_c = yy_meta[(unsigned int) yy_c];
2033: }
2034: yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
2035: }
2036:
2037: return yy_current_state;
2038: }
2039:
2040: /* yy_try_NUL_trans - try to make a transition on the NUL character
2041: *
2042: * synopsis
2043: * next_state = yy_try_NUL_trans( current_state );

```

```

2044: */
2045: static yy_state_type yy_try_NUL_trans (yy_state_type yy_current_state )
2046: {
2047:     register int yy_is_jam;
2048:     register char *yy_cp = (yy_c_buf_p);
2049:
2050:     register YY_CHAR yy_c = 1;
2051:     if ( yy_accept[yy_current_state] )
2052:     {
2053:         (yy_last_accepting_state) = yy_current_state;
2054:         (yy_last_accepting_cpos) = yy_cp;
2055:     }
2056:     while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
2057:     {
2058:         yy_current_state = (int) yy_def[yy_current_state];
2059:         if ( yy_current_state >= 815 )
2060:             yy_c = yy_meta[(unsigned int) yy_c];
2061:     }
2062:     yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
2063:     yy_is_jam = (yy_current_state == 814);
2064:
2065:     return yy_is_jam ? 0 : yy_current_state;
2066: }
2067:
2068: #ifndef YY_NO_INPUT
2069: #ifdef __cplusplus
2070:     static int yyinput (void)
2071: #else
2072:     static int input (void)
2073: #endif
2074:
2075: {
2076:     int c;
2077:
2078:     *(yy_c_buf_p) = (yy_hold_char);
2079:
2080:     if ( *(yy_c_buf_p) == YY_END_OF_BUFFER_CHAR )
2081:     {
2082:         /* yy_c_buf_p now points to the character we want to return.
2083:          * If this occurs *before* the EOB characters, then it's a
2084:          * valid NUL; if not, then we've hit the end of the buffer.

```

```

2085:      */
2086:      if ( (yy_c_buf_p) < &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] )
2087:          /* This was really a NUL. */
2088:          *(yy_c_buf_p) = '\0';
2089:
2090:      else
2091:          { /* need more input */
2092:              int offset = (yy_c_buf_p) - (yytext_ptr);
2093:              ++(yy_c_buf_p);
2094:
2095:              switch ( yy_get_next_buffer( ) )
2096:              {
2097:                  case EOB_ACT_LAST_MATCH:
2098:                      /* This happens because yy_g_n_b()
2099:                       * sees that we've accumulated a
2100:                       * token and flags that we need to
2101:                       * try matching the token before
2102:                       * proceeding.  But for input(),
2103:                       * there's no matching to consider.
2104:                       * So convert the EOB_ACT_LAST_MATCH
2105:                       * to EOB_ACT_END_OF_FILE.
2106:                       */
2107:
2108:                      /* Reset buffer status. */
2109:                      xTEDSrestart(xTEDSin );
2110:
2111:                      /*FALLTHROUGH*/
2112:
2113:                  case EOB_ACT_END_OF_FILE:
2114:                      {
2115:                          if ( xTEDSwrap( ) )
2116:                              return EOF;
2117:
2118:                          if ( ! (yy.did_buffer_switch_on_eof) )
2119:                              YY_NEW_FILE;
2120:                          #ifdef __cplusplus
2121:                              return yyinput();
2122:                          #else
2123:                              return input();
2124:                          #endif
2125:                      }

```

```

2126:
2127:         case EOB_ACT_CONTINUE_SCAN:
2128:             (yy_c_buf_p) = (yytext_ptr) + offset;
2129:             break;
2130:         }
2131:     }
2132: }
2133:
2134: c = *(unsigned char *) (yy_c_buf_p);  /* cast for 8-bit char's */
2135: *(yy_c_buf_p) = '\0';  /* preserve xTEDStext */
2136: (yy_hold_char) = *++(yy_c_buf_p);
2137:
2138: return c;
2139: }
2140: #endif  /* ifndef YY_NO_INPUT */
2141:
2142: /** Immediately switch to a different input stream.
2143:  * @param input_file A readable stream.
2144:  *
2145:  * @note This function does not reset the start condition to @c INITIAL .
2146:  */
2147: void xTEDSrestart (FILE * input_file )
2148: {
2149:
2150: if ( ! YY_CURRENT_BUFFER ){
2151:     xTEDSensure_buffer_stack ();
2152:     YY_CURRENT_BUFFER_LVALUE =
2153:         xTEDS_create_buffer(xTEDSin,YY_BUF_SIZE );
2154: }
2155:
2156: xTEDS_init_buffer(YY_CURRENT_BUFFER,input_file );
2157: xTEDS_load_buffer_state( );
2158: }
2159:
2160: /** Switch to a different input buffer.
2161:  * @param new_buffer The new input buffer.
2162:  *
2163:  */
2164: void xTEDS_switch_to_buffer (YY_BUFFER_STATE new_buffer )
2165: {
2166:

```

```

2167: /* TODO. We should be able to replace this entire function body
2168:  * with
2169:  *      xTEDSpop_buffer_state();
2170:  *      xTEDSpush_buffer_state(new_buffer);
2171:  */
2172: xTEDSensure_buffer_stack ();
2173: if ( YY_CURRENT_BUFFER == new_buffer )
2174:     return;
2175:
2176: if ( YY_CURRENT_BUFFER )
2177:     {
2178:     /* Flush out information for old buffer. */
2179:     *(yy_c_buf_p) = (yy_hold_char);
2180:     YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
2181:     YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
2182:     }
2183:
2184: YY_CURRENT_BUFFER_LVALUE = new_buffer;
2185: xTEDS_load_buffer_state( );
2186:
2187: /* We don't actually know whether we did this switch during
2188:  * EOF (xTEDSwrap()) processing, but the only time this flag
2189:  * is looked at is after xTEDSwrap() is called, so it's safe
2190:  * to go ahead and always set it.
2191:  */
2192: (yy_did_buffer_switch_on_eof) = 1;
2193: }
2194:
2195: static void xTEDS_load_buffer_state (void)
2196: {
2197:     (yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
2198:     (yytext_ptr) = (yy_c_buf_p) = YY_CURRENT_BUFFER_LVALUE->yy_buf_pos;
2199:     xTEDSin = YY_CURRENT_BUFFER_LVALUE->yy_input_file;
2200:     (yy_hold_char) = *(yy_c_buf_p);
2201: }
2202:
2203: /** Allocate and initialize an input buffer state.
2204:  * @param file A readable stream.
2205:  * @param size The character buffer size in bytes. When in doubt, use @c YY_BUF_SIZE.
2206:  *
2207:  * @return the allocated buffer state.

```

```

2208: */
2209: YY_BUFFER_STATE xTEDS_create_buffer (FILE * file, int size )
2210: {
2211:     YY_BUFFER_STATE b;
2212:
2213:     b = (YY_BUFFER_STATE) xTEDSalloc(sizeof( struct yy_buffer_state ) );
2214:     if ( ! b )
2215:         YY_FATAL_ERROR( "out of dynamic memory in xTEDS_create_buffer()" );
2216:
2217:     b->yy_buf_size = size;
2218:
2219:     /* yy_ch_buf has to be 2 characters longer than the size given because
2220:      * we need to put in 2 end-of-buffer characters.
2221:      */
2222:     b->yy_ch_buf = (char *) xTEDSalloc(b->yy_buf_size + 2 );
2223:     if ( ! b->yy_ch_buf )
2224:         YY_FATAL_ERROR( "out of dynamic memory in xTEDS_create_buffer()" );
2225:
2226:     b->yy_is_our_buffer = 1;
2227:
2228:     xTEDS_init_buffer(b,file );
2229:
2230:     return b;
2231: }
2232:
2233: /** Destroy the buffer.
2234:  * @param b a buffer created with xTEDS_create_buffer()
2235:  *
2236:  */
2237: void xTEDS_delete_buffer (YY_BUFFER_STATE b )
2238: {
2239:
2240:     if ( ! b )
2241:         return;
2242:
2243:     if ( b == YY_CURRENT_BUFFER ) /* Not sure if we should pop here. */
2244:         YY_CURRENT_BUFFER_LVALUE = (YY_BUFFER_STATE) 0;
2245:
2246:     if ( b->yy_is_our_buffer )
2247:         xTEDSfree((void *) b->yy_ch_buf );
2248:

```

```

2249: xTEDSfree((void *) b );
2250: }
2251:
2252: #ifndef __cplusplus
2253: extern int isatty (int );
2254: #endif /* __cplusplus */
2255:
2256: /* Initializes or reinitializes a buffer.
2257:  * This function is sometimes called more than once on the same buffer,
2258:  * such as during a xTEDSrestart() or at EOF.
2259:  */
2260: static void xTEDS_init_buffer (YY_BUFFER_STATE b, FILE * file )
2261: {
2262: {
2263: int oerrno = errno;
2264:
2265: xTEDS_flush_buffer(b );
2266:
2267: b->yy_input_file = file;
2268: b->yy_fill_buffer = 1;
2269:
2270: /* If b is the current buffer, then xTEDS_init_buffer was _probably_
2271:  * called from xTEDSrestart() or through yy_get_next_buffer.
2272:  * In that case, we don't want to reset the lineno or column.
2273:  */
2274: if (b != YY_CURRENT_BUFFER){
2275:     b->yy_bs_lineno = 1;
2276:     b->yy_bs_column = 0;
2277: }
2278:
2279: b->yy_is_interactive = file ? (isatty( fileno(file) ) > 0) : 0;
2280:
2281: errno = oerrno;
2282: }
2283:
2284: /** Discard all buffered characters. On the next scan, YY_INPUT will be called.
2285:  * @param b the buffer state to be flushed, usually @c YY_CURRENT_BUFFER.
2286:  *
2287:  */
2288: void xTEDS_flush_buffer (YY_BUFFER_STATE b )
2289: {

```



```

2290:     if ( ! b )
2291:         return;
2292:
2293:     b->yy_n_chars = 0;
2294:
2295:     /* We always need two end-of-buffer characters. The first causes
2296:      * a transition to the end-of-buffer state. The second causes
2297:      * a jam in that state.
2298:      */
2299:     b->yy_ch_buf[0] = YY_END_OF_BUFFER_CHAR;
2300:     b->yy_ch_buf[1] = YY_END_OF_BUFFER_CHAR;
2301:
2302:     b->yy_buf_pos = &b->yy_ch_buf[0];
2303:
2304:     b->yy_at_bol = 1;
2305:     b->yy_buffer_status = YY_BUFFER_NEW;
2306:
2307:     if ( b == YY_CURRENT_BUFFER )
2308:         xTEDS_load_buffer_state( );
2309: }
2310:
2311: /** Pushes the new state onto the stack. The new state becomes
2312:  * the current state. This function will allocate the stack
2313:  * if necessary.
2314:  * @param new_buffer The new state.
2315:  *
2316:  */
2317: void xTEDSpush_buffer_state (YY_BUFFER_STATE new_buffer )
2318: {
2319:     if (new_buffer == NULL)
2320:         return;
2321:
2322:     xTEDSensure_buffer_stack();
2323:
2324:     /* This block is copied from xTEDS_switch_to_buffer. */
2325:     if ( YY_CURRENT_BUFFER )
2326:     {
2327:         /* Flush out information for old buffer. */
2328:         *(yy_c_buf_p) = (yy_hold_char);
2329:         YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
2330:         YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);

```

```

2331:     }
2332:
2333:  /* Only push if top exists. Otherwise, replace top. */
2334:  if (YY_CURRENT_BUFFER)
2335:      (yy_buffer_stack_top)++;
2336:  YY_CURRENT_BUFFER_LVALUE = new_buffer;
2337:
2338:  /* copied from xTEDS_switch_to_buffer. */
2339:  xTEDS_load_buffer_state( );
2340:  (yy_did_buffer_switch_on_eof) = 1;
2341: }
2342:
2343: /** Removes and deletes the top of the stack, if present.
2344:  * The next element becomes the new top.
2345:  *
2346:  */
2347: void xTEDSpop_buffer_state (void)
2348: {
2349:     if (!YY_CURRENT_BUFFER)
2350:         return;
2351:
2352:     xTEDS_delete_buffer(YY_CURRENT_BUFFER );
2353:     YY_CURRENT_BUFFER_LVALUE = NULL;
2354:     if ((yy_buffer_stack_top) > 0)
2355:         --(yy_buffer_stack_top);
2356:
2357:     if (YY_CURRENT_BUFFER) {
2358:         xTEDS_load_buffer_state( );
2359:         (yy_did_buffer_switch_on_eof) = 1;
2360:     }
2361: }
2362:
2363: /* Allocates the stack if it does not exist.
2364:  * Guarantees space for at least one push.
2365:  */
2366: static void xTEDSensure_buffer_stack (void)
2367: {
2368:     int num_to_alloc;
2369:
2370:     if (!(yy_buffer_stack)) {
2371:

```

```

2372:    /* First allocation is just for 2 elements, since we don't know if this
2373:     * scanner will even need a stack. We use 2 instead of 1 to avoid an
2374:     * immediate realloc on the next call.
2375:     */
2376:    num_to_alloc = 1;
2377:    (yy_buffer_stack) = (struct yy_buffer_state**)xTEDSalloc
2378:        (num_to_alloc * sizeof(struct yy_buffer_state*)
2379:        );
2380:
2381:    memset((yy_buffer_stack), 0, num_to_alloc * sizeof(struct yy_buffer_state*));
2382:
2383:    (yy_buffer_stack_max) = num_to_alloc;
2384:    (yy_buffer_stack_top) = 0;
2385:    return;
2386: }
2387:
2388: if ((yy_buffer_stack_top) >= ((yy_buffer_stack_max)) - 1){
2389:
2390:     /* Increase the buffer to prepare for a possible push. */
2391:     int grow_size = 8 /* arbitrary grow size */;
2392:
2393:     num_to_alloc = (yy_buffer_stack_max) + grow_size;
2394:     (yy_buffer_stack) = (struct yy_buffer_state**)xTEDSrealloc
2395:         ((yy_buffer_stack),
2396:         num_to_alloc * sizeof(struct yy_buffer_state*)
2397:         );
2398:
2399:     /* zero only the new slots.*/
2400:     memset((yy_buffer_stack) + (yy_buffer_stack_max), 0, grow_size * sizeof(struct
yy_buffer_state*));
2401:     (yy_buffer_stack_max) = num_to_alloc;
2402: }
2403: }
2404:
2405: /** Setup the input buffer state to scan directly from a user-specified character buffer.
2406:  * @param base the character buffer
2407:  * @param size the size in bytes of the character buffer
2408:  *
2409:  * @return the newly allocated buffer state object.
2410:  */
2411: YY_BUFFER_STATE xTEDS_scan_buffer (char * base, yy_size_t size )

```

```

2412: {
2413:   YY_BUFFER_STATE b;
2414:
2415:   if ( size < 2 ||
2416:       base[size-2] != YY_END_OF_BUFFER_CHAR ||
2417:       base[size-1] != YY_END_OF_BUFFER_CHAR )
2418:     /* They forgot to leave room for the EOB's. */
2419:     return 0;
2420:
2421:   b = (YY_BUFFER_STATE) xTEDSalloc(sizeof( struct yy_buffer_state ) );
2422:   if ( ! b )
2423:     YY_FATAL_ERROR( "out of dynamic memory in xTEDS_scan_buffer()" );
2424:
2425:   b->yy_buf_size = size - 2; /* "- 2" to take care of EOB's */
2426:   b->yy_buf_pos = b->yy_ch_buf = base;
2427:   b->yy_is_our_buffer = 0;
2428:   b->yy_input_file = 0;
2429:   b->yy_n_chars = b->yy_buf_size;
2430:   b->yy_is_interactive = 0;
2431:   b->yy_at_bol = 1;
2432:   b->yy_fill_buffer = 0;
2433:   b->yy_buffer_status = YY_BUFFER_NEW;
2434:
2435:   xTEDS_switch_to_buffer(b );
2436:
2437:   return b;
2438: }
2439:
2440: /** Setup the input buffer state to scan a string. The next call to xTEDSlex() will
2441:  * scan from a @e copy of @a str.
2442:  * @param yystr a NUL-terminated string to scan
2443:  *
2444:  * @return the newly allocated buffer state object.
2445:  * @note If you want to scan bytes that may contain NUL values, then use
2446:  *       xTEDS_scan_bytes() instead.
2447:  */
2448: YY_BUFFER_STATE xTEDS_scan_string (yyconst char * yystr )
2449: {
2450:
2451:   return xTEDS_scan_bytes(yystr,strlen(yystr) );
2452: }

```

```

2453:
2454: /** Setup the input buffer state to scan the given bytes. The next call to xTEDSlex() will
2455:  * scan from a @e copy of @a bytes.
2456:  * @param bytes the byte buffer to scan
2457:  * @param len the number of bytes in the buffer pointed to by @a bytes.
2458:  *
2459:  * @return the newly allocated buffer state object.
2460:  */
2461: YY_BUFFER_STATE xTEDS_scan_bytes (yyconst char * yybytes, int _yybytes_len )
2462: {
2463:   YY_BUFFER_STATE b;
2464:   char *buf;
2465:   yy_size_t n;
2466:   int i;
2467:
2468:   /* Get memory for full buffer, including space for trailing EOB's. */
2469:   n = _yybytes_len + 2;
2470:   buf = (char *) xTEDSalloc(n );
2471:   if ( ! buf )
2472:     YY_FATAL_ERROR( "out of dynamic memory in xTEDS_scan_bytes()" );
2473:
2474:   for ( i = 0; i < _yybytes_len; ++i )
2475:     buf[i] = yybytes[i];
2476:
2477:   buf[_yybytes_len] = buf[_yybytes_len+1] = YY_END_OF_BUFFER_CHAR;
2478:
2479:   b = xTEDS_scan_buffer(buf,n );
2480:   if ( ! b )
2481:     YY_FATAL_ERROR( "bad buffer in xTEDS_scan_bytes()" );
2482:
2483:   /* It's okay to grow etc. this buffer, and we should throw it
2484:    * away when we're done.
2485:    */
2486:   b->yy_is_our_buffer = 1;
2487:
2488:   return b;
2489: }
2490:
2491: #ifndef YY_EXIT_FAILURE
2492: #define YY_EXIT_FAILURE 2
2493: #endif

```

```

2494:
2495: static void yy_fatal_error (yyconst char* msg )
2496: {
2497:     (void) fprintf( stderr, "%s \n", msg );
2498:     exit( YY_EXIT_FAILURE );
2499: }
2500:
2501: /* Redefine yyless() so it works in section 3 code. */
2502:
2503: #undef yyless
2504: #define yyless(n) \
2505:     do \
2506:     { \
2507:         /* Undo effects of setting up xTEDStext. */ \
2508:         int yyless_macro_arg = (n); \
2509:         YY_LESS_LINENO(yyless_macro_arg); \
2510:         xTEDStext[xTEDSleng] = (yy_hold_char); \
2511:         (yy_c_buf_p) = xTEDStext + yyless_macro_arg; \
2512:         (yy_hold_char) = *(yy_c_buf_p); \
2513:         *(yy_c_buf_p) = '\0'; \
2514:         xTEDSleng = yyless_macro_arg; \
2515:     } \
2516:     while ( 0 )
2517:
2518: /* Accessor methods (get/set functions) to struct members. */
2519:
2520: /** Get the current line number.
2521:  *
2522:  */
2523: int xTEDSget_lineno (void)
2524: {
2525:
2526:     return xTEDSlineno;
2527: }
2528:
2529: /** Get the input stream.
2530:  *
2531:  */
2532: FILE *xTEDSget_in (void)
2533: {
2534:     return xTEDSin;

```

```

2535: }
2536:
2537: /** Get the output stream.
2538: *
2539: */
2540: FILE *xTEDSget_out (void)
2541: {
2542:     return xTEDSout;
2543: }
2544:
2545: /** Get the length of the current token.
2546: *
2547: */
2548: int xTEDSget_leng (void)
2549: {
2550:     return xTEDSleng;
2551: }
2552:
2553: /** Get the current token.
2554: *
2555: */
2556:
2557: char *xTEDSget_text (void)
2558: {
2559:     return xTEDStext;
2560: }
2561:
2562: /** Set the current line number.
2563: * @param line_number
2564: *
2565: */
2566: void xTEDSset_lineno (int line_number )
2567: {
2568:
2569:     xTEDSlineno = line_number;
2570: }
2571:
2572: /** Set the input stream. This does not discard the current
2573: * input buffer.
2574: * @param in_str A readable stream.
2575: *

```

```

2576: * @see xTEDS_switch_to_buffer
2577: */
2578: void xTEDSset_in (FILE * in_str )
2579: {
2580:     xTEDSin = in_str ;
2581: }
2582:
2583: void xTEDSset_out (FILE * out_str )
2584: {
2585:     xTEDSout = out_str ;
2586: }
2587:
2588: int xTEDSget_debug (void)
2589: {
2590:     return xTEDS_flex_debug;
2591: }
2592:
2593: void xTEDSset_debug (int bdebug )
2594: {
2595:     xTEDS_flex_debug = bdebug ;
2596: }
2597:
2598: static int yy_init_globals (void)
2599: {
2600:     /* Initialization is the same as for the non-reentrant scanner.
2601:      * This function is called from xTEDSlex_destroy(), so don't allocate here.
2602:      */
2603:
2604:     (yy_buffer_stack) = 0;
2605:     (yy_buffer_stack_top) = 0;
2606:     (yy_buffer_stack_max) = 0;
2607:     (yy_c_buf_p) = (char *) 0;
2608:     (yy_init) = 0;
2609:     (yy_start) = 0;
2610:
2611:     /* Defined in main.c */
2612:     #ifdef YY_STDINIT
2613:         xTEDSin = stdin;
2614:         xTEDSout = stdout;
2615:     #else
2616:         xTEDSin = (FILE *) 0;

```



```

2617:  xTEDSout = (FILE *) 0;
2618: #endif
2619:
2620:  /* For future reference: Set errno on error, since we are called by
2621:   * xTEDSlex_init()
2622:   */
2623:  return 0;
2624: }
2625:
2626: /* xTEDSlex_destroy is for both reentrant and non-reentrant scanners. */
2627: int xTEDSlex_destroy (void)
2628: {
2629:
2630:  /* Pop the buffer stack, destroying each element. */
2631:  while(YY_CURRENT_BUFFER){
2632:      xTEDS_delete_buffer(YY_CURRENT_BUFFER );
2633:      YY_CURRENT_BUFFER_LVALUE = NULL;
2634:      xTEDSpop_buffer_state();
2635:  }
2636:
2637:  /* Destroy the stack itself. */
2638:  xTEDSfree((yy_buffer_stack) );
2639:  (yy_buffer_stack) = NULL;
2640:
2641:  /* Reset the globals. This is important in a non-reentrant scanner so the next time
2642:   * xTEDSlex() is called, initialization will occur. */
2643:  yy_init_globals( );
2644:
2645:  return 0;
2646: }
2647:
2648: /*
2649:  * Internal utility routines.
2650:  */
2651:
2652: #ifndef yytext_ptr
2653: static void yy_flex_strncpy (char* s1, yyconst char * s2, int n )
2654: {
2655:     register int i;
2656:     for ( i = 0; i < n; ++i )
2657:         s1[i] = s2[i];

```

```

2658: }
2659: #endif
2660:
2661: #ifdef YY_NEED_STRLEN
2662: static int yy_flex_strlen (yyconst char * s )
2663: {
2664:     register int n;
2665:     for ( n = 0; s[n]; ++n )
2666:         ;
2667:
2668:     return n;
2669: }
2670: #endif
2671:
2672: void *xTEDSalloc (yy_size_t size )
2673: {
2674:     return (void *) malloc( size );
2675: }
2676:
2677: void *xTEDSrealloc (void * ptr, yy_size_t size )
2678: {
2679:     /* The cast to (char *) in the following accommodates both
2680:      * implementations that use char* generic pointers, and those
2681:      * that use void* generic pointers. It works with the latter
2682:      * because both ANSI C and C++ allow castless assignment from
2683:      * any pointer type to void*, and deal with argument conversions
2684:      * as though doing an assignment.
2685:      */
2686:     return (void *) realloc( (char *) ptr, size );
2687: }
2688:
2689: void xTEDSfree (void * ptr )
2690: {
2691:     free( (char *) ptr ); /* see xTEDSrealloc() for (char *) cast */
2692: }
2693:
2694: #define YYTABLES_NAME "yytables"
2695:
2696: #line 147 "xTEDS.l"
2697:
2698:

```

```
2699:
2700: int xTEDSwrap() {return 1;}
2701: void xTEDSError(char *s)
2702: {
2703:  printf("  Syntax error in xTEDS on line %d at token  \"%s \". \n",line_count,xTEDStext);
2704: }
2705:
```

File: sdm/common/xTEDS/xTEDSVariable.h

```
1: #ifndef __SDM_XTEDS_VARIABLE_H_
2: #define __SDM_XTEDS_VARIABLE_H_
3:
4: #include "xTEDSItem.h"
5: #include "SDMDataTypes.h"
6: #include "xTEDSDrange.h"
7: #include "xTEDSCurve.h"
8: #include "xTEDSLocation.h"
9: #include "xTEDSOrientationList.h"
10: #include "VariableDef.h"
11: #include "xTEDSQualifier.h"
12: extern "C"
13: {
14: #include "xTEDSParser.h"
15: }
16:
17: #define ATTR_INIT_VALUE  0xFFFFFFFF
18:
19: class xTEDSVariable:public xTEDSItem
20: {
21: public:
22:  /** default ctor
23:   *
24:   * Default ctor sets 0/Null values for all except data Format is SDM_UNIT16 and Length of 1.
25:   */
26:  xTEDSVariable();
27:
28:  /** dtor
29:   */
30:  virtual ~xTEDSVariable();
31:
32:  /** @brief */
33:  virtual MessageDef* RegInfo() const;
34:  virtual bool RegInfoMatch(const char* name, const xTEDSQualifierList&, const char* interface)
35:  const;
36:
37:  void VarRegInfo(char* InfoBufferOut, int BufferSize, int& Offset) const;
38:  void VarRef(char* InfoBufferOut, int BufferSize) const;
39:  /**
```

```

39:  */
40: void SetVariable(const variable* NewVar);
41: VariableDef* VarInfo() const;
42:
43: virtual void PrintDebug() const;
44:
45: //-----
46: // Getter/Setter Methods added for testability
47: //-----
48:
49: void setDataFormat(SDMDDataTypes& newDataFormat);
50: const SDMDDataTypes& getDataFormat() const;
51:
52: void setLength(int newLength);
53: int getLength() const;
54:
55: void setKind(const char* newKind);
56: const char* getKind() const;
57:
58: void setQualifier(const char* newQualifier);
59: const char* getQualifier() const;
60:
61: void setID(int newID);
62: int getID() const;
63:
64: void setRangeMin(const char* newRangeMin);
65: const char* getRangeMin() const;
66:
67: void setRangeMax(const char* newRangeMax);
68: const char* getRangeMax() const;
69:
70: void setDefaultValue(const char* newDefaultValue);
71: const char* getDefaultValue() const;
72:
73: void setPrecision(unsigned int newPrecesion);
74: unsigned int getPrecision() const;
75:
76: void setUnits(const char* newUnits);
77: const char* getUnits() const;
78:
79: void setAccuracy(const char* newAccuracy);

```

```

80: const char* getAccuracy() const;
81:
82: void setScaleFactor(const char* newScaleFactor);
83: const char* getScaleFactor() const;
84:
85: void setScaleUnits(const char* newScaleUnits);
86: const char* getScaleUnits() const;
87:
88: void setRLow(const char* newRLow);
89: const char* getRLow() const;
90:
91: void setRHigh(const char* newRHigh);
92: const char* getRHigh() const;
93:
94: void setYLow(const char* newYLow);
95: const char* getYLow() const;
96:
97: void setYHigh(const char* newYHigh);
98: const char* getYHigh() const;
99:
100: void setInvalidValue(const char* NewVal);
101: const char* getInvalidValue() const;
102:
103: private:
104: /**
105:  * Copy ctor - not defined to prevent default object copy semantics
106:  */
107: xTEDSVariable(const xTEDSVariable&);
108:
109: /**
110:  * Assignment operator - not defined to prevent default object copy semantics
111:  */
112: xTEDSVariable& operator=(const xTEDSVariable&);
113:
114: virtual bool MatchesQualifier(const xTEDSQualifierList&) const;
115: bool MatchesQualifier(xTEDSQualifierList*);
116: char* VarInfoRequest() const;
117: void getFormat(char* strFormatOut, int BufSize) const;
118:
119: void setLocation(const location *loc);
120: void setOrientation(const orientation *orient);

```

```

121: void setDRange(const drange* range);
122: void setCurve(const curve* curves);
123:
124: /**
125:  * Helper for setting string members...
126:  */
127: void setStringMember( char* & memberStrVar, const char* newValue);
128:
129: //-----
130: // Data members
131: //-----
132:
133: SDMDDataTypes m_dataFormat;
134: int m_iLength;
135: char* m_strKind;
136: char* m_strQualifier;    // NOTE: Variable qualifier has been deprecated, to be removed in next
xTEDS schema release
137: int m_iID;               // NOTE: Variable ID has been deprecated, to be removed in next xTEDS
schema release
138: char* m_strRangeMin;
139: char* m_strRangeMax;
140: char* m_strDefaultValue;
141: unsigned int m_iPrecision;
142: char* m_strUnits;
143: char* m_strAccuracy;
144: char* m_strScaleFactor;
145: char* m_strScaleUnits;
146: xTEDSLocation *m_varLocation;
147: xTEDSOrientationList *m_varOrientation;
148: xTEDSDrange *m_varDrange;
149: xTEDSCurve *m_varCurve;
150: char* m_strRLow;
151: char* m_strRHigh;
152: char* m_strYLow;
153: char* m_strYHigh;
154: char* m_strInvalidValue;
155:
156: };
157:
158: #endif

```

File: sdm/common/xTEDS/xTEDSFaultMsg.h

```
1: #ifndef __SDM_XTEDS_FAULT_MSG_H_
2: #define __SDM_XTEDS_FAULT_MSG_H_
3:
4: #include "xTEDSMessage.h"
5: #include "xTEDSVariable.h"
6: #include "xTEDSVariableList.h"
7: #include "MessageDef.h"
8:
9: class xTEDSFaultMsg:public xTEDSMessage
10: {
11: public:
12: xTEDSFaultMsg();
13: virtual ~xTEDSFaultMsg();
14:
15: MessageDef* RegInfo(void) const;
16: virtual bool RegInfoMatch(const char* name, const xTEDSQualifierList&, const char* interface)
    const;
17: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
    Interface) const;
18: bool SetFaultMsg(const fault_msg* FaultMsg, const xTEDSVariableList& VariablesList);
19:
20: virtual void PrintDebug() const;
21: private:
22: virtual bool MatchesQualifier(const xTEDSQualifierList&) const;
23: void xTEDSInfo(char* InfoBufferOut, int BufferSize) const;
24: };
25:
26: #endif
```


File: sdm/common/xTEDS/MessageDef.h

```
1: #ifndef __SDM_MESSAGE_DEF_H
2: #define __SDM_MESSAGE_DEF_H
3: #include "../sdmLib.h"
4:
5: #include "../message_defs.h"
6: #include "../message/SDMMessage_ID.h"
7:
8: class SDMLIB_API MessageDef
9: {
10: public:
11:     MessageDef();
12:     MessageDef(const MessageDef&);
13:     ~MessageDef();
14:
15:     MessageDef& operator=(const MessageDef&);
16:
17:     void operator += (MessageDef* Right);
18:
19:     void Print();
20:     void Join(MessageDef*);
21:     void SetInterfaceMessageID(const SDMMessage_ID& );
22:     SDMMessage_ID GetInterfaceMessageID() const { return mMessageInterfaceID; }
23:     void SetDefinitions(const char* NewDefinitions);
24:     void SetxTEDSPortion(const char* NewxTEDS);
25:     char* GetxTEDSPortion() const { return xTEDSPortion; }
26:     char* GetDefinitions() const { return def; }
27: private:
28:     SDMMessage_ID mMessageInterfaceID;
29:     char* def;
30:     char* xTEDSPortion;
31: public: //TODO
32:     MessageDef* next;
33:
34: };
35:
36:
37: #endif
```

File: sdm/common/xTEDS/xTEDSWrapper.cpp

```
1: #include <string.h>
2: #include "xTEDSWrapper.h"
3:
4: xTEDSWrapper::xTEDSWrapper():Type(WRAPPER_EMPTY),InterfaceName(NULL),InterfaceID(-
1)
5: {
6: }
7:
8: xTEDSWrapper::~xTEDSWrapper()
9: {
10: if (InterfaceName != NULL)
11:     free (InterfaceName);
12: }
13:
14: void xTEDSWrapper::setInterfaceName(const char* Name)
15: {
16: if (InterfaceName != NULL)
17:     free(InterfaceName);
18: InterfaceName = strdup(Name);
19: }
20:
21: void xTEDSWrapper::setInterfaceID(int ID)
22: {
23: InterfaceID = ID;
24: }
25:
26: #ifndef REMOVE_DEBUG_OUTPUT
27: void xTEDSWrapper::PrintDebug() const
28: {
29: char TypeStr[128];
30:
31: switch(Type)
32: {
33:     case WRAPPER_NOTIFICATION:
34:         strcpy(TypeStr, "WRAPPER_NOTIFICATION");
35:         break;
36:     case WRAPPER_REQUEST:
37:         strcpy(TypeStr, "WRAPPER_REQUEST");
38:         break;
```

```
39:     case WRAPPER_COMMAND:
40:         strcpy(TypeStr, "WRAPPER_COMMAND");
41:         break;
42:     case WRAPPER_EMPTY:
43:         strcpy(TypeStr, "WRAPPER_EMPTY");
44:         break;
45:     default:
46:         break;
47: }
48: printf("xTEDSWrapper interface %s id %d type %s \n", InterfaceName, InterfaceID, TypeStr);
49: }
50: #endif
```

File: sdm/common/xTEDS/xTEDSDataMsg.h

```
1: #ifndef __SDM_XTEDS_DATA_MSG_H_
2: #define __SDM_XTEDS_DATA_MSG_H_
3:
4: #include "xTEDSMessage.h"
5: #include "xTEDSVariable.h"
6: #include "SDMDataRates.h"
7: #include "MessageDef.h"
8: #include "xTEDSVariableList.h"
9: extern "C"
10: {
11: #include "xTEDSParser.h"
12: }
13:
14: class xTEDSDataMsg:public xTEDSMessage
15: {
16: public:
17: xTEDSDataMsg();
18: virtual ~xTEDSDataMsg();
19:
20: MessageDef* RegInfo(void) const;
21: virtual bool RegInfoMatch(const char* name, const xTEDSQualifierList&, const char* interface)
const;
22: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
Interface) const;
23: bool SetDataMsg(const data_msg* NewDataMsg, const xTEDSVariableList& VariablesList);
24:
25: SDMDataRates msg_arrival;
26: float msg_rate;
27:
28: virtual void PrintDebug() const;
29: private:
30: virtual bool MatchesQualifier(const xTEDSQualifierList&) const;
31: void xTEDSInfo(char* InfoBufferOut, int BufferSize) const;
32: };
33:
34: #endif
```

File: sdm/common/xTEDS/xTEDS.tab.h

```
1: /* A Bison parser, made by GNU Bison 1.875d. */
2:
3: /* Skeleton parser for Yacc-like parsing with Bison,
4:  Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.
5:
6:  This program is free software; you can redistribute it and/or modify
7:  it under the terms of the GNU General Public License as published by
8:  the Free Software Foundation; either version 2, or (at your option)
9:  any later version.
10:
11:  This program is distributed in the hope that it will be useful,
12:  but WITHOUT ANY WARRANTY; without even the implied warranty of
13:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14:  GNU General Public License for more details.
15:
16:  You should have received a copy of the GNU General Public License
17:  along with this program; if not, write to the Free Software
18:  Foundation, Inc., 59 Temple Place - Suite 330,
19:  Boston, MA 02111-1307, USA. */
20:
21: /* As a special exception, when this file is copied by Bison into a
22:  Bison output file, you may use that output file without restriction.
23:  This special exception was added by the Free Software Foundation
24:  in version 1.24 of Bison. */
25:
26: /* Tokens. */
27: #ifndef YYTOKENTYPE
28: # define YYTOKENTYPE
29:  /* Put the tokens into the symbol table, so that GDB and other debuggers
30:   know about them. */
31:  enum yytokentype {
32:    EQUAL_SY = 258,
33:    CLOSE_SY = 259,
34:    SLASHCLOSE_SY = 260,
35:    OPEN_XML_SY = 261,
36:    CLOSE_xTEDS_SY = 262,
37:    OPEN_xTEDS_SY = 263,
38:    OPEN_APP_SY = 264,
39:    OPEN_VAR_SY = 265,
```

40: CLOSE_VAR_SY = 266,
41: OPEN_DRANGE_SY = 267,
42: CLOSE_DRANGE_SY = 268,
43: OPEN_OPTION_SY = 269,
44: OPEN_CURVE_SY = 270,
45: CLOSE_CURVE_SY = 271,
46: OPEN_COEFF_SY = 272,
47: OPEN_DATA_MSG_SY = 273,
48: CLOSE_DATA_MSG_SY = 274,
49: OPEN_VARIABLE_REF_SY = 275,
50: OPEN_COMMAND_MSG_SY = 276,
51: CLOSE_COMMAND_MSG_SY = 277,
52: NAME_SY = 278,
53: KIND_SY = 279,
54: ID_SY = 280,
55: CLOSE_ORIENTATION_SY = 281,
56: QUALIFIER_SY = 282,
57: DESCRIPTION_SY = 283,
58: MANUFACTURER_ID_SY = 284,
59: VERSION_SY = 285,
60: MODEL_ID_SY = 286,
61: VERSION_LETTER_SY = 287,
62: SERIAL_NUMBER_SY = 288,
63: CALIBRATION_DATE_SY = 289,
64: SENSITIVITY_AT_REF_SY = 290,
65: REF_FREQ_SY = 291,
66: REF_TEMP_SY = 292,
67: MEASUREMENT_RANGE_SY = 293,
68: ELECTRICAL_OUTPUT_SY = 294,
69: QUALITY_FACTOR_SY = 295,
70: TEMP_COEFF_SY = 296,
71: DIRECTION_XYZ_SY = 297,
72: CAL_DUE_DATE_SY = 298,
73: POWER_REQS_SY = 299,
74: VALUE_SY = 300,
75: ALARM_SY = 301,
76: MSG_ARRIVAL_SY = 302,
77: MSG_RATE_SY = 303,
78: STRING = 304,
79: FLOAT = 305,
80: INT = 306,

81: PRECISION_SY = 307,
82: RANGE_MAX_SY = 308,
83: CLOSE_LOCATION_SY = 309,
84: FORMAT_SY = 310,
85: ACCURACY_SY = 311,
86: RANGE_MIN_SY = 312,
87: SCALE_FACTOR_SY = 313,
88: UNITS_SY = 314,
89: DEFAULT_VALUE_SY = 315,
90: OPEN_DEVICE_SY = 316,
91: SCALE_UNITS_SY = 317,
92: LENGTH_SY = 318,
93: EXPONENT_SY = 319,
94: SCHEMA_LOCATION_SY = 320,
95: XMLNS_SY = 321,
96: XMLNS_XSI_SY = 322,
97: CLOSE_OPTION_SY = 323,
98: OPEN_INTERFACE_SY = 324,
99: OPEN_COMMAND_SY = 325,
100: OPEN_NOTIFICATION_SY = 326,
101: OPEN_REQUEST_SY = 327,
102: OPEN_FAULT_MSG_SY = 328,
103: COMPONENT_KEY_SY = 329,
104: SPA_U_HUB_SY = 330,
105: SPA_U_PORT_SY = 331,
106: EXTENDS_SY = 332,
107: CLOSE_COMMAND_SY = 333,
108: CLOSE_NOTIFICATION_SY = 334,
109: CLOSE_REQUEST_SY = 335,
110: CLOSE_FAULT_MSG_SY = 336,
111: OPEN_QUALIFIER_SY = 337,
112: CLOSE_QUALIFIER_SY = 338,
113: CLOSE_APP_SY = 339,
114: CLOSE_DEVICE_SY = 340,
115: CLOSE_INTERFACE_SY = 341,
116: MEMORY_MINIMUM_SY = 342,
117: OPERATING_SYSTEM_SY = 343,
118: PATH_FOR_ASSEMBLY_SY = 344,
119: PATH_ON_SPACECRAFT_SY = 345,
120: X_SY = 346,
121: Y_SY = 347,

```

122:  Z_SY = 348,
123:  AXIS_SY = 349,
124:  ANGLE_SY = 350,
125:  OPEN_LOCATION_SY = 351,
126:  OPEN_ORIENTATION_SY = 352,
127:  CLOSE_XML_SY = 353,
128:  ENCODING_SY = 354,
129:  STANDALONE_SY = 355,
130:  CLOSE_VARIABLE_REF_SY = 356,
131:  CLOSE_COEFF_SY = 357,
132:  R_LOW_SY = 358,
133:  R_HIGH_SY = 359,
134:  Y_LOW_SY = 360,
135:  Y_HIGH_SY = 361,
136:  INVALID_VALUE_SY = 362,
137:  BAD_TERMINAL_SY = 363
138:  };
139: #endif
140: #define EQUAL_SY 258
141: #define CLOSE_SY 259
142: #define SLASHCLOSE_SY 260
143: #define OPEN_XML_SY 261
144: #define CLOSE_xTEDS_SY 262
145: #define OPEN_xTEDS_SY 263
146: #define OPEN_APP_SY 264
147: #define OPEN_VAR_SY 265
148: #define CLOSE_VAR_SY 266
149: #define OPEN_DRANGE_SY 267
150: #define CLOSE_DRANGE_SY 268
151: #define OPEN_OPTION_SY 269
152: #define OPEN_CURVE_SY 270
153: #define CLOSE_CURVE_SY 271
154: #define OPEN_COEFF_SY 272
155: #define OPEN_DATA_MSG_SY 273
156: #define CLOSE_DATA_MSG_SY 274
157: #define OPEN_VARIABLE_REF_SY 275
158: #define OPEN_COMMAND_MSG_SY 276
159: #define CLOSE_COMMAND_MSG_SY 277
160: #define NAME_SY 278
161: #define KIND_SY 279
162: #define ID_SY 280

```


163: #define CLOSE_ORIENTATION_SY 281
164: #define QUALIFIER_SY 282
165: #define DESCRIPTION_SY 283
166: #define MANUFACTURER_ID_SY 284
167: #define VERSION_SY 285
168: #define MODEL_ID_SY 286
169: #define VERSION_LETTER_SY 287
170: #define SERIAL_NUMBER_SY 288
171: #define CALIBRATION_DATE_SY 289
172: #define SENSITIVITY_AT_REF_SY 290
173: #define REF_FREQ_SY 291
174: #define REF_TEMP_SY 292
175: #define MEASUREMENT_RANGE_SY 293
176: #define ELECTRICAL_OUTPUT_SY 294
177: #define QUALITY_FACTOR_SY 295
178: #define TEMP_COEFF_SY 296
179: #define DIRECTION_XYZ_SY 297
180: #define CAL_DUE_DATE_SY 298
181: #define POWER_REQS_SY 299
182: #define VALUE_SY 300
183: #define ALARM_SY 301
184: #define MSG_ARRIVAL_SY 302
185: #define MSG_RATE_SY 303
186: #define STRING 304
187: #define FLOAT 305
188: #define INT 306
189: #define PRECISION_SY 307
190: #define RANGE_MAX_SY 308
191: #define CLOSE_LOCATION_SY 309
192: #define FORMAT_SY 310
193: #define ACCURACY_SY 311
194: #define RANGE_MIN_SY 312
195: #define SCALE_FACTOR_SY 313
196: #define UNITS_SY 314
197: #define DEFAULT_VALUE_SY 315
198: #define OPEN_DEVICE_SY 316
199: #define SCALE_UNITS_SY 317
200: #define LENGTH_SY 318
201: #define EXPONENT_SY 319
202: #define SCHEMA_LOCATION_SY 320
203: #define XMLNS_SY 321

204: #define XMLNS_XSI_SY 322
205: #define CLOSE_OPTION_SY 323
206: #define OPEN_INTERFACE_SY 324
207: #define OPEN_COMMAND_SY 325
208: #define OPEN_NOTIFICATION_SY 326
209: #define OPEN_REQUEST_SY 327
210: #define OPEN_FAULT_MSG_SY 328
211: #define COMPONENT_KEY_SY 329
212: #define SPA_U_HUB_SY 330
213: #define SPA_U_PORT_SY 331
214: #define EXTENDS_SY 332
215: #define CLOSE_COMMAND_SY 333
216: #define CLOSE_NOTIFICATION_SY 334
217: #define CLOSE_REQUEST_SY 335
218: #define CLOSE_FAULT_MSG_SY 336
219: #define OPEN_QUALIFIER_SY 337
220: #define CLOSE_QUALIFIER_SY 338
221: #define CLOSE_APP_SY 339
222: #define CLOSE_DEVICE_SY 340
223: #define CLOSE_INTERFACE_SY 341
224: #define MEMORY_MINIMUM_SY 342
225: #define OPERATING_SYSTEM_SY 343
226: #define PATH_FOR_ASSEMBLY_SY 344
227: #define PATH_ON_SPACECRAFT_SY 345
228: #define X_SY 346
229: #define Y_SY 347
230: #define Z_SY 348
231: #define AXIS_SY 349
232: #define ANGLE_SY 350
233: #define OPEN_LOCATION_SY 351
234: #define OPEN_ORIENTATION_SY 352
235: #define CLOSE_XML_SY 353
236: #define ENCODING_SY 354
237: #define STANDALONE_SY 355
238: #define CLOSE_VARIABLE_REF_SY 356
239: #define CLOSE_COEFF_SY 357
240: #define R_LOW_SY 358
241: #define R_HIGH_SY 359
242: #define Y_LOW_SY 360
243: #define Y_HIGH_SY 361
244: #define INVALID_VALUE_SY 362

```

245: #define BAD_TERMINAL_SY 363
246:
247:
248:
249:
250: #if ! defined (YYSTYPE) && ! defined (YYSTYPE_IS_DECLARED)
251: #line 38 "xTEDS.y"
252: typedef union YYSTYPE {
253:     int integer;
254:     float real;
255:     char* str;
256:     struct variable_data* var;
257:     struct variable_reference* var_ref;
258:     struct qualifier_data* qual;
259:     struct coefficient_data* coef;
260:     struct curve_data* curve;
261:     struct option_data* curveoption;
262:     struct drange_data* drange;
263:     struct location_data* location;
264:     struct orientation_data* orientation;
265:     struct fault_message* fault_msg;
266:     struct data_message* data_msg;
267:     struct command_message* cmd_msg;
268:     struct command_type* command;
269:     struct notification_type* notification;
270:     struct request_type* request;
271:     struct message_type* message;
272:     struct interface_type* interface;
273:     struct xteds* xteds;
274:     struct app_device_attributes* attr;
275: } YYSTYPE;
276: /* Line 1285 of yacc.c. */
277: #line 278 "xTEDS.tab.h"
278: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
279: # define YYSTYPE_IS_DECLARED 1
280: # define YYSTYPE_IS_TRIVIAL 1
281: #endif
282:
283: extern YYSTYPE xTEDSlval;
284:
285:

```

286:

File: sdm/common/xTEDS/xTEDSLocation.h

```
1: #ifndef __SDM_XTEDS_LOCATION_H_
2: #define __SDM_XTEDS_LOCATION_H_
3:
4: extern "C"
5: {
6: #include "xTEDSParser.h"
7: }
8:
9: class xTEDSLocation
10: {
11: public:
12: xTEDSLocation();
13: ~xTEDSLocation();
14: xTEDSLocation(const xTEDSLocation&);
15: xTEDSLocation& operator=(const xTEDSLocation&);
16:
17: void setX(const char *xStr);
18: void setY(const char *yStr);
19: void setZ(const char *zStr);
20: void setUnits(const char *unitsStr);
21:
22: void setLocation(const char *x, const char *y, const char *z, const char *units);
23: void setLocation(const location *loc);
24:
25: void VarInfoRequest(char* InfoBufferOut, size_t BufferSize);
26: private:
27: char *m_strX;
28: char *m_strY;
29: char *m_strZ;
30: char *m_strUnits;
31: };
32:
33: #endif
```

File: sdm/common/xTEDS/xTEDSCurve.cpp

```
1: #include "xTEDSCurve.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10: xTEDSCurve::xTEDSCurve():m_strName(NULL),m_strDescription(NULL),m_xclCoefs()
11: {
12: }
13:
14:          /*xTEDSCurve::xTEDSCurve(const          xTEDSCurve&
b):m_strName(NULL),m_strDescription(NULL),m_xclCoefs()
15: {
16: if(m_strName!=NULL) free(m_strName);
17: m_strName = strdup(b.m_strName);
18: if(m_strDescription!=NULL) free(m_strDescription);
19: m_strDescription = strdup(b.m_strDescription);
20: //if(coefs!=NULL) free(coefs);
21: //coefs = strdup(b.coefs);
22: }
23:
24: xTEDSCurve& xTEDSCurve::operator=(const xTEDSCurve& b)
25: {
26: if(m_strName!=NULL) free(m_strName);
27: m_strName = strdup(b.m_strName);
28: if(m_strDescription!=NULL) free(m_strDescription);
29: m_strDescription = strdup(b.m_strDescription);
30: //if(coefs!=NULL) free(coefs);
31: //coefs = strdup(b.coefs);
32: return *this;
33: }*/
34:
35: xTEDSCurve::~xTEDSCurve()
36: {
37: if(m_strName!=NULL) free(m_strName);
38: if(m_strDescription!=NULL) free(m_strDescription);
```

```

39: }
40:
41: void xTEDSCurve::setCurve(const curve* NewCurve)
42: {
43: if (NewCurve == NULL) return;
44: setName(NewCurve->name);
45: setDescription(NewCurve->description);
46: m_xclCoefs.setCoefList(NewCurve->coefs);
47: }
48:
49: void xTEDSCurve::setName(const char* new_name)
50: {
51: if (new_name == NULL) return;
52: if(m_strName!=NULL) free(m_strName);
53: m_strName = strdup(new_name);
54: }
55:
56: void xTEDSCurve::setDescription(const char* new_description)
57: {
58: if (new_description == NULL) return;
59: if(m_strDescription!=NULL) free(m_strDescription);
60: m_strDescription = strdup(new_description);
61: }
62:
63: void xTEDSCurve::VarInfoRequest( char* InfoBufferOut, size_t BufferSize ) const
64: {
65: char Buf[MSG_DEF_SIZE];
66:
67: if (m_strName == NULL)
68:     return ;
69:
70: if(m_strDescription != NULL)
71: {
72:     snprintf(Buf, sizeof(Buf), "<Curve name= \"%s \" description= \"%s \"",m_strName,
m_strDescription);
73: }
74: else
75: {
76:     snprintf(Buf, sizeof(Buf), "<Curve name= \"%s \"", m_strName);
77: }
78: if(m_xclCoefs.IsEmpty())

```

```
79: {
80:     strncat(Buf, ">", sizeof(Buf) - strlen(Buf));
81: }
82: else
83: {
84:     strncat(Buf, ">", sizeof(Buf) - strlen(Buf));
85:
86:     size_t CurBufLength = strlen(Buf);
87:     m_xclCoefs.VarInfoRequest(Buf + CurBufLength, sizeof(Buf) - CurBufLength);
88:
89:     strncat(Buf, " \n \t</Curve>", sizeof(Buf) - strlen(Buf));
90: }
91: strncat(InfoBufferOut, Buf, BufferSize - 1);
92: }
93:
```


File: sdm/common/xTEDS/xTEDSNotification.h

```
1: #ifndef __SDM_XTEDS_NOTIFICATION_H_
2: #define __SDM_XTEDS_NOTIFICATION_H_
3:
4: #include "xTEDSItem.h"
5: #include "xTEDSDataMsg.h"
6: #include "xTEDSFaultMsg.h"
7: #include "xTEDSWrapper.h"
8: #include "xTEDSVariableList.h"
9:
10: class xTEDSNotification:public xTEDSWrapper
11: {
12: public:
13: xTEDSNotification();
14: xTEDSNotification(const xTEDSNotification&);
15: virtual ~xTEDSNotification();
16:
17: virtual MessageDef* RegInfo(const char* ItemName) const;
18: virtual bool RegInfoMatch(const char* Name, const xTEDSQualifierList& , const char* Interface)
const;
19: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
Interface) const;
20: bool SetNotification(const notification* Notification, const xTEDSVariableList& VariablesList);
21: bool ContainsMessage(const SDMMMessage_ID& MessageID) const;
22: SDMMMessage_ID GetFaultMessageID() const;
23:
24: xTEDSNotification& operator=(const xTEDSNotification&);
25: virtual void PrintDebug() const;
26: private:
27: xTEDSDataMsg* m_DataMessage;
28: xTEDSFaultMsg* m_FaultMessage;
29: };
30:
31: #endif
```

File: sdm/common/xTEDS/xTEDSSubscriptionList.h

```
1: #ifndef _SDM_XTEDS_SUBSCRIPTION_LIST_H_
2: #define _SDM_XTEDS_SUBSCRIPTION_LIST_H_
3:
4: #include "../message/SDMMessage_ID.h"
5: #include "../message/SDMComponent_ID.h"
6:
7: class xTEDSSubscriptionListNode
8: {
9: public:
10: xTEDSSubscriptionListNode() : pSub(NULL), pNext(NULL) {}
11: xTEDSSubscription* pSub;
12: xTEDSSubscriptionListNode* pNext;
13: };
14:
15: class xTEDSSubscriptionList
16: {
17: public:
18: void Add(const SDMComponent_ID& idSubscriber, const SDMMessage_ID& idMessage);
19: void Remove(const SDMComponent_ID& idSubscriber, const SDMMessage_ID& idMessage);
20: private:
21: bool Contains(const SDMComponent_ID& idSubscriber, const SDMMessage_ID& idMessage);
22: };
23:
24:
25: #endif
26:
```

File: sdm/common/xTEDS/xTEDSParser.c

```
1: #include "xTEDSParser.h"
2: #include <stdlib.h>
3: #include <string.h>
4: #include <stdio.h>
5:
6: xteds* result;
7: variable* var_result;
8: /*
9:  README:
10:
11:  This source file contains functions to build an xTEDS tree as the parser encounters the text.
12:  The functions are, in general, separated into new_*, link_*, merge_*, delete_*, and add_*
13:  names.
14:
15:  The new_* functions are used to allocate a structure type and initialize all pointer members to
16:  NULL. Also other member types should be initialized to some start value. Note that all new_*
17:  functions dynamically allocate the structure, and should be deleted using the corresponding
18:  delete_* function.
19:
20:  The link_* functions are used to link items of a list. Many of the structure types have members
21:  called "next" which refers to a link list. The link functions are used to link the instances.
22:
23:  The merge_* functions are used to add attributes and arbitrarily ordered elements to the structure.
24:  Notice that on many of the members of a structure, within these functions, there is a "else if"
25:  clause that prevents a memory leak that occurs if some attribute is specified in the xTEDS multiple
26:  times. This cannot be caught by the parser, and the funtions will simply throw away the most recent
27:  addition.
28:
29:  The delete_* functions free the dynamically allocated memory allocated by the new_* functions. If
30:  the structure member is a linked list, the delete function should recursively delete the list.
31:
32:  The add_* functions are used to simply add a pointer to a list member of a structure.
33:
34: */
35: qualifier_type* link_qualifiers(qualifier_type* a, qualifier_type* b)
36: {
37:     qualifier_type* cur;
38:
39:     if(a==NULL) return b;
```

```

40: if(b==NULL) return a;
41:
42: for(cur=a;cur->next!=NULL;cur=cur->next);
43: cur->next = b;
44: return a;
45: }
46:
47: qualifier_type* merge_qualifiers(qualifier_type* a,qualifier_type* b)
48: {
49: if(a==NULL) return b;
50: if(b==NULL) return a;
51:
52: if(a->name==NULL) a->name = b->name;
53:  else if(b->name!=NULL) free(b->name);
54:
55: if(a->value==NULL) a->value = b->value;
56:  else if(b->value!=NULL) free(b->value);
57:
58: if(a->units==NULL) a->units = b->units;
59:  else if(b->units!=NULL) free(b->units);
60:
61: free(b);
62: return a;
63: }
64:
65: qualifier_type* new_qualifier()
66: {
67: qualifier_type* temp;
68:
69: temp = (qualifier_type*)malloc(sizeof(qualifier_type));
70: temp->name= NULL;
71: temp->value= NULL;
72: temp->units= NULL;
73: temp->next= NULL;
74:
75: return temp;
76: }
77:
78: coef* new_coef()
79: {
80: coef* temp;

```

```

81:
82: temp = (coef*)malloc(sizeof(coef));
83: temp->exponent= NULL;
84: temp->value= NULL;
85: temp->description= NULL;
86: temp->next= NULL;
87:
88: return temp;
89: }
90:
91: coef* link_coefs(coef* a,coef* b)
92: {
93: coef* cur;
94:
95: if(a==NULL) return b;
96: if(b==NULL) return a;
97:
98: for(cur=a;cur->next!=NULL;cur=cur->next);
99: cur->next = b;
100: return a;
101: }
102:
103: coef* merge_coefs(coef* a,coef* b)
104: {
105: if(a==NULL) return b;
106: if(b==NULL) return a;
107:
108: if(a->exponent==NULL) a->exponent = b->exponent;
109: else if(b->exponent!=NULL) free(b->exponent);
110:
111: if(a->value==NULL) a->value = b->value;
112: else if(b->value!=NULL) free(b->value);
113:
114: if(a->description==NULL) a->description = b->description;
115: else if(b->description!=NULL) free(b->description);
116:
117: free(b);
118: return a;
119: }
120:
121: curve* new_curve()

```

```

122: {
123:     curve* temp;
124:
125:     temp = (curve*)malloc(sizeof(curve));
126:     temp->name= NULL;
127:     temp->description= NULL;
128:     temp->coefs= NULL;
129:     return temp;
130: }
131:
132: curve* merge_curves(curve* a, curve* b)
133: {
134:     if(a==NULL) return b;
135:     if(b==NULL) return a;
136:
137:     if(a->name==NULL) a->name = b->name;
138:     else if(b->name!=NULL) free(b->name);
139:
140:     if(a->description==NULL) a->description = b->description;
141:     else if(b->description!=NULL) free(b->description);
142:
143:     if(a->coefs==NULL) a->coefs = b->coefs;
144:     else if(b->coefs!=NULL) delete_coef(b->coefs);
145:
146:     free(b);
147:     return a;
148: }
149:
150: curve* curve_add_coefs(curve* curve_type, coef* coefs)
151: {
152:     if(curve_type==NULL) return curve_type;
153:     curve_type->coefs = coefs;
154:     return curve_type;
155: }
156:
157: curveoption* new_option()
158: {
159:     curveoption* temp;
160:
161:     temp = (curveoption*)malloc(sizeof(curveoption));
162:     temp->name= NULL;

```

```

163:   temp->value= NULL;
164:   temp->description= NULL;
165:   temp->alarm= NULL;
166:   temp->next= NULL;
167:
168:   return temp;
169: }
170:
171: curveoption* link_options(curveoption* a,curveoption* b)
172: {
173:   curveoption* cur;
174:
175:   if(a==NULL) return b;
176:   if(b==NULL) return a;
177:
178:   for(cur=a;cur->next!=NULL;cur=cur->next);
179:   cur->next = b;
180:   return a;
181: }
182:
183: curveoption* merge_options(curveoption* a,curveoption* b)
184: {
185:   if(a==NULL) return b;
186:   if(b==NULL) return a;
187:
188:   if(a->name==NULL) a->name = b->name;
189:   else if(b->name!=NULL) free(b->name);
190:
191:   if(a->value==NULL) a->value = b->value;
192:   else if(b->value!=NULL) free(b->value);
193:
194:   if(a->description==NULL) a->description = b->description;
195:   else if(b->description!=NULL) free(b->description);
196:
197:   if(a->alarm==NULL) a->alarm = b->alarm;
198:   else if(b->alarm!=NULL) free(b->alarm);
199:
200:   free(b);
201:   return a;
202: }
203:

```

```

204: drange* new_drange()
205: {
206:     drange* temp;
207:
208:     temp = (drange*)malloc(sizeof(drange));
209:     temp->name= NULL;
210:     temp->description= NULL;
211:     temp->options= NULL;
212:     return temp;
213: }
214:
215: drange* merge_dranges(drange* a,drange* b)
216: {
217:     if(a==NULL) return b;
218:     if(b==NULL) return a;
219:
220:     if(a->name==NULL) a->name = b->name;
221:     else if(b->name!=NULL) free(b->name);
222:
223:     if(a->description==NULL) a->description = b->description;
224:     else if(b->description!=NULL) free(b->description);
225:
226:     if(a->options==NULL) a->options = b->options;
227:     else if(b->options!=NULL) delete_option(b->options);
228:
229:     free(b);
230:     return a;
231: }
232:
233: drange* drange_add_options(drange* drange_type, curveoption* options)
234: {
235:     if(drange_type==NULL) return drange_type;
236:     drange_type->options = options;
237:     return drange_type;
238: }
239:
240: location* new_location()
241: {
242:     location* temp;
243:
244:     temp = (location*)malloc(sizeof(location));

```



```

245:   temp->x_value = NULL;
246:   temp->y_value = NULL;
247:   temp->z_value = NULL;
248:   temp->units = NULL;
249:   return temp;
250: }
251:
252: location* merge_locations(location* a,location* b)
253: {
254:     if (a == NULL) return b;
255:     if (b == NULL) return a;
256:
257:     if (a->x_value == NULL) a->x_value = b->x_value;
258:     else if (b->x_value != NULL) free(b->x_value);
259:
260:     if (a->y_value == NULL) a->y_value = b->y_value;
261:     else if (b->y_value != NULL) free(b->y_value);
262:
263:     if (a->z_value == NULL) a->z_value = b->z_value;
264:     else if (b->z_value != NULL) free(b->z_value);
265:
266:     if (a->units == NULL) a->units = b->units;
267:     else if (b->units != NULL) free(b->units);
268:
269:     free(b);
270:     return a;
271: }
272:
273: orientation* link_orientations(orientation* a, orientation* b)
274: {
275:     orientation* cur;
276:
277:     if (a == NULL) return b;
278:     if (b == NULL) return a;
279:
280:     for(cur=a; cur->next!=NULL; cur=cur->next);
281:     cur->next = b;
282:     return a;
283: }
284:
285: orientation* merge_orientations(orientation* a, orientation* b)

```

```

286: {
287:   if (a == NULL) return b;
288:   if (b == NULL) return a;
289:
290:   if (a->axis_value == NULL) a->axis_value = b->axis_value;
291:   else if(b->axis_value != NULL) free(b->axis_value);
292:
293:   if (a->angle_value == NULL) a->angle_value = b->angle_value;
294:   else if(b->angle_value != NULL) free(b->angle_value);
295:
296:   if (a->units_value == NULL) a->units_value = b->units_value;
297:   else if (b->units_value != NULL) free(b->units_value);
298:
299:   free (b);
300:   return a;
301: }
302:
303: orientation* new_orientation()
304: {
305:   orientation* temp;
306:
307:   temp = (orientation*)malloc(sizeof(orientation));
308:   temp->axis_value = NULL;
309:   temp->angle_value = NULL;
310:   temp->units_value = NULL;
311:   temp->next = NULL;
312:   return temp;
313: }
314:
315: variable* link_variables(variable* a,variable* b)
316: {
317:   variable* cur;
318:
319:   if(a==NULL) return b;
320:   if(b==NULL) return a;
321:
322:   for(cur=a;cur->next!=NULL;cur=cur->next);
323:   cur->next = b;
324:   return a;
325: }
326:

```

```

327: variable* merge_variables(variable* a,variable* b)
328: {
329:     if(a==NULL) return b;
330:     if(b==NULL) return a;
331:     /* The else if blocks are required to prevent a possible memory leak occurring when a attribute
332:        in the xTEDS is specified multiple times, this isn't really enforceable in the xTEDS schema
333:        unless an exhaustive list of all permutations of attribute orders was specified in the grammar*/
334:     if(a->length==NULL) a->length = b->length;
335:     else if(b->length!=NULL) free(b->length);
336:
337:     if(a->kind==NULL) a->kind = b->kind;
338:     else if(b->kind!=NULL) free(b->kind);
339:
340:     if(a->name==NULL) a->name = b->name;
341:     else if(b->name!=NULL) free(b->name);
342:
343:     if(a->qualifier==NULL) a->qualifier = b->qualifier;
344:     else if(b->qualifier!=NULL) free(b->qualifier);
345:
346:     if(a->id==NULL) a->id = b->id;
347:     else if(b->id!=NULL) free(b->id);
348:
349:     if(a->range_min==NULL) a->range_min = b->range_min;
350:     else if(b->range_min!=NULL) free(b->range_min);
351:
352:     if(a->range_max==NULL) a->range_max = b->range_max;
353:     else if(b->range_max!=NULL) free(b->range_max);
354:
355:     if(a->default_value==NULL) a->default_value = b->default_value;
356:     else if(b->default_value!=NULL) free(b->default_value);
357:
358:     if(a->precision==NULL) a->precision = b->precision;
359:     else if(b->precision!=NULL) free(b->precision);
360:
361:     if(a->units==NULL) a->units = b->units;
362:     else if(b->units!=NULL) free(b->units);
363:
364:     if(a->accuracy==NULL) a->accuracy = b->accuracy;
365:     else if(b->accuracy!=NULL) free(b->accuracy);
366:
367:     if(a->scale_factor==NULL) a->scale_factor = b->scale_factor;

```

```

368:     else if(b->scale_factor!=NULL) free(b->scale_factor);
369:
370:     if(a->scale_units==NULL) a->scale_units = b->scale_units;
371:     else if(b->scale_units!=NULL) free(b->scale_units);
372:
373:     if(a->format==NULL) a->format = b->format;
374:     else if(b->format!=NULL) free(b->format);
375:
376:     if(a->description==NULL) a->description = b->description;
377:     else if(b->description!=NULL) free(b->description);
378:
379:     if(a->qualifiers==NULL) a->qualifiers = b->qualifiers;
380:     else if(b->qualifiers!=NULL) free(b->qualifiers);
381:
382:     if(a->curves==NULL) a->curves = b->curves;
383:     else if(b->curves!=NULL) delete_curve(b->curves);
384:
385:     if(a->dranges==NULL) a->dranges = b->dranges;
386:     else if(b->dranges!=NULL) delete_drangle(b->dranges);
387:
388:     if(a->location_data==NULL) a->location_data = b->location_data;
389:     else if(b->location_data!=NULL) delete_location(b->location_data);
390:
391:     if (a->r_low == NULL) a->r_low = b->r_low;
392:     else if(b->r_low!=NULL) free(b->r_low);
393:
394:     if (a->r_high == NULL) a->r_high = b->r_high;
395:     else if(b->r_high!=NULL) free(b->r_high);
396:
397:     if (a->y_low == NULL) a->y_low = b->y_low;
398:     else if(b->y_low!=NULL) free(b->y_low);
399:
400:     if (a->y_high == NULL) a->y_high = b->y_high;
401:     else if(b->y_high!=NULL) free(b->y_high);
402:
403:     if (a->invalid_value == NULL) a->invalid_value = b->invalid_value;
404:     else if(b->invalid_value!=NULL) free(b->invalid_value);
405:
406:     a->orientation_data = link_orientations(a->orientation_data,b->orientation_data);
407:     free(b);
408:     /*Keep a reference to this variable tree for VarInfoParser*/

```

```

409:   var_result = a;
410:   return a;
411: }
412:
413: variable* new_variable()
414: {
415:     variable* temp;
416:
417:     temp = (variable*)malloc(sizeof(variable));
418:     temp->length = NULL;
419:     temp->kind= NULL;
420:     temp->name= NULL;
421:     temp->qualifier= NULL;
422:     temp->id= NULL;
423:     temp->range_min= NULL;
424:     temp->range_max= NULL;
425:     temp->default_value= NULL;
426:     temp->precision= NULL;
427:     temp->units= NULL;
428:     temp->accuracy= NULL;
429:     temp->scale_factor= NULL;
430:     temp->scale_units= NULL;
431:     temp->format= NULL;
432:     temp->description= NULL;
433:     temp->qualifiers= NULL;
434:     temp->curves= NULL;
435:     temp->dranges= NULL;
436:     temp->next= NULL;
437:     temp->interface_name= NULL;
438:     temp->interface_id= NULL;
439:     temp->location_data = NULL;
440:     temp->orientation_data = NULL;
441:     temp->r_low = NULL;
442:     temp->r_high = NULL;
443:     temp->y_low = NULL;
444:     temp->y_high = NULL;
445:     temp->invalid_value = NULL;
446:     return temp;
447: }
448:
449: var_ref* link_variable_ref(var_ref* a,var_ref* b)

```

```

450: {
451:     var_ref* cur;
452:
453:     if(a==NULL) return b;
454:     if(b==NULL) return a;
455:
456:     for(cur=a;cur->next!=NULL;cur=cur->next);
457:     cur->next = b;
458:     return a;
459: }
460:
461: var_ref* new_variable_ref(char* name)
462: {
463:     var_ref* temp;
464:
465:     temp = (var_ref*)malloc(sizeof(var_ref));
466:     temp->name = name;
467:     temp->next = NULL;
468:     return temp;
469: }
470:
471: fault_msg* merge_fault_msg(fault_msg* a,fault_msg* b)
472: {
473:     if(a==NULL) return b;
474:     if(b==NULL) return a;
475:
476:     if(a->name==NULL) a->name = b->name;
477:     else if(b->name!=NULL) free(b->name);
478:
479:     if(a->id==NULL) a->id = b->id;
480:     else if(b->id!=NULL) free(b->id);
481:
482:     if(a->description==NULL) a->description = b->description;
483:     else if(b->description!=NULL) free(b->description);
484:
485:     if(a->variables==NULL) a->variables = b->variables;
486:     if(a->qualifiers==NULL) a->qualifiers = b->qualifiers;
487:     free(b);
488:     return a;
489: }
490:

```

```

491: fault_msg* new_fault_msg()
492: {
493:     fault_msg* temp;
494:     temp = (fault_msg*)malloc(sizeof(fault_msg));
495:     temp->name= NULL;
496:     temp->id= NULL;
497:     temp->description= NULL;
498:     temp->variables= NULL;
499:     temp->qualifiers= NULL;
500:     temp->interface_name= NULL;
501:     temp->interface_id= NULL;
502:     return temp;
503: }
504:
505: fault_msg* fault_add_var_refs(fault_msg* msg,var_ref* ref)
506: {
507:     if(msg==NULL) return msg;
508:     msg->variables = ref;
509:     return msg;
510: }
511:
512: data_msg* link_data_msg(data_msg* a,data_msg* b)
513: {
514:     data_msg* cur;
515:
516:     if(a==NULL) return b;
517:     if(b==NULL) return a;
518:
519:     for(cur=a;cur->next!=NULL;cur=cur->next);
520:     cur->next = b;
521:     return a;
522: }
523:
524: data_msg* merge_data_msg(data_msg* a,data_msg* b)
525: {
526:     if(a==NULL) return b;
527:     if(b==NULL) return a;
528:
529:     if(a->name==NULL) a->name = b->name;
530:     else if(b->name!=NULL) free(b->name);
531:

```

```

532:   if(a->id==NULL) a->id = b->id;
533:   else if(b->id!=NULL) free(b->id);
534:
535:   if(a->description==NULL) a->description = b->description;
536:   else if(b->description!=NULL) free(b->description);
537:
538:   if(a->msg_arrival==NULL) a->msg_arrival = b->msg_arrival;
539:   else if(b->msg_arrival!=NULL) free(b->msg_arrival);
540:
541:   if(a->msg_rate==NULL) a->msg_rate = b->msg_rate;
542:   else if(b->msg_rate!=NULL) free(b->msg_rate);
543:
544:   if(a->qualifiers==NULL) a->qualifiers = b->qualifiers;
545:   if(a->variables==NULL) a->variables = b->variables;
546:   free(b);
547:   return a;
548: }
549:
550: data_msg* new_data_msg()
551: {
552:   data_msg* temp;
553:   temp = (data_msg*)malloc(sizeof(data_msg));
554:   temp->name= NULL;
555:   temp->id= NULL;
556:   temp->description= NULL;
557:   temp->msg_arrival=NULL;
558:   temp->msg_rate=NULL;
559:   temp->variables=NULL;
560:   temp->qualifiers=NULL;
561:   temp->next= NULL;
562:   temp->interface_name= NULL;
563:   temp->interface_id= NULL;
564:   return temp;
565: }
566:
567: data_msg* data_add_var_refs(data_msg* msg,var_ref* ref)
568: {
569:   if(msg==NULL) return msg;
570:   msg->variables = ref;
571:   return msg;
572: }

```



```

573:
574: cmd_msg* link_cmd_msg(cmd_msg* a,cmd_msg* b)
575: {
576:     cmd_msg* cur;
577:
578:     if(a==NULL) return b;
579:     if(b==NULL) return a;
580:
581:     for(cur=a;cur->next!=NULL;cur=cur->next);
582:     cur->next = b;
583:     return a;
584: }
585:
586: cmd_msg* merge_cmd_msg(cmd_msg* a,cmd_msg* b)
587: {
588:     if(a==NULL) return b;
589:     if(b==NULL) return a;
590:
591:     if(a->name==NULL) a->name = b->name;
592:     else if(b->name!=NULL) free(b->name);
593:
594:     if(a->id==NULL) a->id = b->id;
595:     else if(b->id!=NULL) free(b->id);
596:
597:     if(a->description==NULL) a->description = b->description;
598:     else if(b->description!=NULL) free(b->description);
599:
600:     if(a->qualifiers==NULL) a->qualifiers = b->qualifiers;
601:     if(a->variables==NULL) a->variables = b->variables;
602:     free(b);
603:     return a;
604: }
605:
606: cmd_msg* new_cmd_msg()
607: {
608:     cmd_msg* temp;
609:     temp = (cmd_msg*)malloc(sizeof(cmd_msg));
610:     temp->name= NULL;
611:     temp->id= NULL;
612:     temp->description= NULL;
613:     temp->variables=NULL;

```

```

614:   temp->qualifiers=NULL;
615:   temp->next= NULL;
616:   temp->interface_name= NULL;
617:   temp->interface_id= NULL;
618:   return temp;
619: }
620:
621: cmd_msg* cmd_add_var_refs(cmd_msg* msg,var_ref* ref)
622: {
623:   if(msg==NULL) return msg;
624:   msg->variables = ref;
625:   return msg;
626: }
627:
628: command* new_command()
629: {
630:   command* temp;
631:   temp = (command*)malloc(sizeof(command));
632:   temp->command_message= NULL;
633:   temp->fault_message= NULL;
634:   temp->next= NULL;
635:   temp->interface_name= NULL;
636:   temp->interface_id= NULL;
637:   return temp;
638: }
639:
640: command* command_add_cmd_msg(command* a ,cmd_msg* b)
641: {
642:   if(a==NULL) return a;
643:   a->command_message = b;
644:   return a;
645: }
646:
647: command* command_add_fault_msg(command* a,fault_msg* b)
648: {
649:   if(a==NULL) return a;
650:   a->fault_message = b;
651:   return a;
652: }
653:
654: command* link_command(command* a, command* b)

```

```

655: {
656:     command* cur;
657:
658:     if(a==NULL) return b;
659:     if(b==NULL) return a;
660:
661:     for(cur=a;cur->next!=NULL;cur=cur->next);
662:     cur->next = b;
663:     return a;
664: }
665:
666: notification* new_notification()
667: {
668:     notification* temp;
669:     temp = (notification*)malloc(sizeof(notification));
670:     temp->data_message= NULL;
671:     temp->fault_message= NULL;
672:     temp->next= NULL;
673:     temp->interface_name= NULL;
674:     temp->interface_id= NULL;
675:     return temp;
676: }
677:
678: notification* notification_add_data_msg(notification* a,data_msg* b)
679: {
680:     if(a==NULL) return a;
681:     a->data_message = b;
682:     return a;
683: }
684:
685: notification* notification_add_fault_msg(notification* a,fault_msg* b)
686: {
687:     if(a==NULL) return a;
688:     a->fault_message = b;
689:     return a;
690: }
691:
692: notification* link_notification(notification* a, notification* b)
693: {
694:     notification* cur;
695:

```

```

696:    if(a==NULL) return b;
697:    if(b==NULL) return a;
698:
699:    for(cur=a;cur->next!=NULL;cur=cur->next);
700:    cur->next = b;
701:    return a;
702: }
703:
704: request* new_request()
705: {
706:     request* temp;
707:     temp = (request*)malloc(sizeof(request));
708:     temp->command_message= NULL;
709:     temp->data_message= NULL;
710:     temp->fault_message= NULL;
711:     temp->next= NULL;
712:     temp->interface_name= NULL;
713:     temp->interface_id= NULL;
714:     return temp;
715: }
716:
717: request* request_add_cmd_msg(request* a,cmd_msg* b)
718: {
719:     if(a==NULL) return a;
720:     a->command_message = b;
721:     return a;
722: }
723:
724: request* request_add_data_msg(request* a,data_msg* b)
725: {
726:     if(a==NULL) return a;
727:     a->data_message = b;
728:     return a;
729: }
730:
731: request* request_add_fault_msg(request* a,fault_msg* b)
732: {
733:     if(a==NULL) return a;
734:     a->fault_message = b;
735:     return a;
736: }

```

```

737:
738: request* link_request(request* a, request* b)
739: {
740:     request* cur;
741:
742:     if(a==NULL) return b;
743:     if(b==NULL) return a;
744:
745:     for(cur=a;cur->next!=NULL;cur=cur->next);
746:     cur->next = b;
747:     return a;
748: }
749:
750: message* new_message()
751: {
752:     message* temp;
753:     temp = (message*)malloc(sizeof(message));
754:     temp->commands= NULL;
755:     temp->notifications= NULL;
756:     temp->requests= NULL;
757:     return temp;
758: }
759:
760: message* message_link_command(message* msg,command* cmd)
761: {
762:     if(msg==NULL) return msg;
763:     msg->commands = link_command(msg->commands,cmd);
764:     return msg;
765: }
766:
767: message* message_link_notification(message* msg,notification* not)
768: {
769:     if(msg==NULL) return msg;
770:     msg->notifications = link_notification(msg->notifications,not);
771:     return msg;
772: }
773:
774: message* message_link_request(message* msg,request* req)
775: {
776:     if(msg==NULL) return msg;
777:     msg->requests = link_request(msg->requests,req);

```

```

778:    return msg;
779: }
780:
781: message* merge_message(message* a,message* b)
782: {
783:     if(a==NULL) return b;
784:     if(b==NULL) return a;
785:     if(a->commands==NULL) a->commands = b->commands;
786:     else a->commands = link_command(a->commands,b->commands);
787:     if(a->notifications==NULL) a->notifications = b->notifications;
788:     else a->notifications = link_notification(a->notifications,b->notifications);
789:     if(a->requests==NULL) a->requests = b->requests;
790:     else a->requests = link_request(a->requests,b->requests);
791:     free(b);
792:     return a;
793: }
794:
795: interface* new_interface()
796: {
797:     interface* temp;
798:     temp = (interface*)malloc(sizeof(interface));
799:     temp->name= NULL;
800:     temp->extends= NULL;
801:     temp->id= NULL;
802:     temp->description= NULL;
803:     temp->qualifiers= NULL;
804:     temp->variables= NULL;
805:     temp->commands= NULL;
806:     temp->notifications= NULL;
807:     temp->requests= NULL;
808:     temp->next= NULL;
809:
810:     return temp;
811: }
812:
813: interface* link_interface(interface* a,interface* b)
814: {
815:     interface* cur;
816:
817:     if(a==NULL) return b;
818:     if(b==NULL) return a;

```

```

819:
820:   for(cur=a;cur->next!=NULL;cur=cur->next);
821:   cur->next = b;
822:   return a;
823: }
824:
825: interface* merge_interface(interface* a,interface* b)
826: {
827:   if(a==NULL) return b;
828:   if(b==NULL) return a;
829:
830:   if(a->name==NULL) a->name = b->name;
831:   else if(b->name!=NULL) free(b->name);
832:
833:   if(a->extends==NULL) a->extends = b->extends;
834:   else if(b->extends!=NULL) free(b->extends);
835:
836:   if(a->id==NULL) a->id = b->id;
837:   else if(b->id!=NULL) free(b->id);
838:
839:   if(a->description==NULL) a->description = b->description;
840:   else if(b->description!=NULL) free(b->description);
841:
842:   /* Note: the below fields are not added using merge_interface, so it is not possible for
843:    both "a" and "b" to contain definitions of the below, so the else clause isn't needed*/
844:   if(a->qualifiers==NULL) a->qualifiers = b->qualifiers;
845:   if(a->variables==NULL) a->variables = b->variables;
846:   if(a->commands==NULL) a->commands = b->commands;
847:   if(a->notifications==NULL) a->notifications = b->notifications;
848:   if(a->requests==NULL) a->requests = b->requests;
849:   free(b);
850:   return a;
851: }
852:
853: interface* interface_add_references(interface* a,qualifier_type* q,variable* v,message* m)
854: {
855:   variable* cur_var = NULL;
856:   command* cur_cmd = NULL;
857:   notification* cur_not = NULL;
858:   request* cur_request = NULL;
859:   if(a==NULL) return a;

```

```

860: a->qualifiers = q;
861: a->variables = v;
862: if(a->id==NULL)
863: {
864:     printf("Error: Interface is missing the id field!! \n");
865:     a->id = strdup("0");
866: }
867: for(cur_var=a->variables;cur_var!=NULL;cur_var=cur_var->next)
868: {
869:     if(a->name != NULL)
870:         cur_var->interface_name = strdup(a->name);
871:     if(a->id != NULL)
872:         cur_var->interface_id = strdup(a->id);
873: }
874: if(m != NULL)
875: {
876:     a->commands = m->commands;
877:     for(cur_cmd=a->commands;cur_cmd!=NULL;cur_cmd=cur_cmd->next)
878:     {
879:         if(a->name != NULL)
880:             cur_cmd->interface_name = strdup(a->name);
881:         if(a->id != NULL)
882:             cur_cmd->interface_id = strdup(a->id);
883:         if(cur_cmd->command_message != NULL)
884:         {
885:             if(a->name != NULL)
886:                 cur_cmd->command_message->interface_name = strdup(a->name);
887:             if(a->id != NULL)
888:                 cur_cmd->command_message->interface_id = strdup(a->id);
889:         }
890:         else
891:             printf("Error: <Command> wrapper is missing a command message definition. \n");
892:         if(cur_cmd->fault_message!=NULL)
893:         {
894:             cur_cmd->fault_message->interface_name = strdup(a->name);
895:             cur_cmd->fault_message->interface_id = strdup(a->id);
896:         }
897:     }
898:     a->notifications = m->notifications;
899:     for(cur_not=a->notifications;cur_not!=NULL;cur_not=cur_not->next)
900:     {

```



```

901:     if(a->name != NULL)
902:         cur_not->interface_name = strdup(a->name);
903:     if(a->id != NULL)
904:         cur_not->interface_id = strdup(a->id);
905:     if(cur_not->data_message != NULL)
906:     {
907:         if(a->name != NULL)
908:             cur_not->data_message->interface_name = strdup(a->name);
909:         if(a->id != NULL)
910:             cur_not->data_message->interface_id = strdup(a->id);
911:     }
912:     else
913:         printf("Error: <Notification> wrapper is missing a data message definition. \n");
914:     if(cur_not->fault_message!=NULL)
915:     {
916:         cur_not->fault_message->interface_name = strdup(a->name);
917:         cur_not->fault_message->interface_id = strdup(a->id);
918:     }
919: }
920: a->requests = m->requests;
921: for(cur_request=a->requests;cur_request!=NULL;cur_request=cur_request->next)
922: {
923:     if(a->name != NULL)
924:         cur_request->interface_name = strdup(a->name);
925:     if(a->id != NULL)
926:         cur_request->interface_id = strdup(a->id);
927:     if(cur_request->command_message != NULL)
928:     {
929:         if(a->name != NULL)
930:             cur_request->command_message->interface_name = strdup(a->name);
931:         if(a->id != NULL)
932:             cur_request->command_message->interface_id = strdup(a->id);
933:     }
934:     else
935:         printf("Error: <Request> wrapper is missing a command message definition. \n");
936:     if(cur_request->data_message != NULL)
937:     {
938:         if(a->name != NULL)
939:             cur_request->data_message->interface_name = strdup(a->name);
940:         if(a->id != NULL)
941:             cur_request->data_message->interface_id = strdup(a->id);

```

```

942:         }
943:     else
944:         printf("Error: <Request> wrapper is missing a data message definition. \n");
945:     if(cur_request->fault_message!=NULL)
946:     {
947:         cur_request->fault_message->interface_name = strdup(a->name);
948:         cur_request->fault_message->interface_id = strdup(a->id);
949:     }
950: }
951: free(m);
952: }
953: return a;
954: }
955:
956: xteds* merge_xteds(xteds* a,xteds* b)
957: {
958:     if(a==NULL) return b;
959:     if(b==NULL) return a;
960:
961:     if(a->name==NULL) a->name = b->name;
962:     if(a->version==NULL) a->version = b->version;
963:     if(a->description==NULL) a->description = b->description;
964:     if(a->xmlns==NULL) a->xmlns = b->xmlns;
965:     if(a->xmlns_xsi==NULL) a->xmlns_xsi = b->xmlns_xsi;
966:     if(a->schema_location==NULL) a->schema_location = b->schema_location;
967:     free(b);
968:     return a;
969: }
970:
971: xteds* new_xteds()
972: {
973:     xteds* temp;
974:     temp = (xteds*)malloc(sizeof(xteds));
975:     temp->name= NULL;
976:     temp->version= NULL;
977:     temp->description= NULL;
978:     temp->xmlns = NULL;
979:     temp->xmlns_xsi = NULL;
980:     temp->schema_location = NULL;
981:     temp->interfaces= NULL;
982:     return temp;

```

```

983: }
984:
985: app_dev_attr* merge_app_dev_attr(app_dev_attr* a,app_dev_attr* b)
986: {
987:     if(a==NULL) return b;
988:     if(b==NULL) return a;
989:
990:     if(a->name==NULL) a->name = b->name;
991:     if(a->kind==NULL) a->kind = b->kind;
992:     if(a->id==NULL) a->id = b->id;
993:     if(a->qualifier==NULL) a->qualifier = b->qualifier;
994:     if(a->description==NULL) a->description = b->description;
995:     if(a->manufacturer_id==NULL) a->manufacturer_id = b->manufacturer_id;
996:     if(a->componentKey==NULL) a->componentKey = b->componentKey;
997:     if(a->model_id==NULL) a->model_id = b->model_id;
998:     if(a->version_letter==NULL) a->version_letter = b->version_letter;
999:     if(a->version==NULL) a->version = b->version;
1000:     if(a->serial_number==NULL) a->serial_number = b->serial_number;
1001:     if(a->calibration_date==NULL) a->calibration_date = b->calibration_date;
1002:     if(a->sensitivity_at_reference==NULL)          a->sensitivity_at_reference      =      b-
>sensitivity_at_reference;
1003:     if(a->reference_frequency==NULL) a->reference_frequency = b->reference_frequency;
1004:     if(a->reference_temperature==NULL) a->reference_temperature = b->reference_temperature;
1005:     if(a->measurement_range==NULL) a->measurement_range = b->measurement_range;
1006:     if(a->electrical_output==NULL) a->electrical_output = b->electrical_output;
1007:     if(a-> quality_factor==NULL) a->quality_factor = b->quality_factor;
1008:     if(a->temperature_coefficient==NULL)          a->temperature_coefficient      =      b-
>temperature_coefficient;
1009:     if(a->direction_xyz==NULL) a->direction_xyz = b->direction_xyz;
1010:     if(a->cal_due_date==NULL) a->cal_due_date = b->cal_due_date;
1011:     if(a->power_requirements==NULL) a->power_requirements = b->power_requirements;
1012:     if(a->spa_u_hub==NULL) a->spa_u_hub = b->spa_u_hub;
1013:     if(a->spa_u_port==NULL) a->spa_u_port = b->spa_u_port;
1014:     a->qualifiers = link_qualifiers(a->qualifiers,b->qualifiers);
1015:     if(a->memory_minimum==NULL) a->memory_minimum = b->memory_minimum;
1016:     if(a->operating_system==NULL) a->operating_system = b->operating_system;
1017:     if(a->path_for_assembly==NULL) a->path_for_assembly = b->path_for_assembly;
1018:     if(a->path_on_spacecraft==NULL) a->path_on_spacecraft = b->path_on_spacecraft;
1019:     free(b);
1020:     return a;
1021: }

```

```

1022:
1023: app_dev_attr* new_app_dev_attr()
1024: {
1025:     app_dev_attr* temp;
1026:     temp = (app_dev_attr*)malloc(sizeof(app_dev_attr));
1027:     temp->name= NULL;
1028:     temp->kind= NULL;
1029:     temp->id= NULL;
1030:     temp->qualifier= NULL;
1031:     temp->description= NULL;
1032:     temp->manufacturer_id= NULL;
1033:     temp->componentKey= NULL;
1034:     temp->model_id= NULL;
1035:     temp->version_letter= NULL;
1036:     temp->version= NULL;
1037:     temp->serial_number= NULL;
1038:     temp->calibration_date= NULL;
1039:     temp->sensitivity_at_reference= NULL;
1040:     temp->reference_frequency= NULL;
1041:     temp->reference_temperature= NULL;
1042:     temp->measurement_range= NULL;
1043:     temp->electrical_output= NULL;
1044:     temp->quality_factor= NULL;
1045:     temp->temperature_coefficient= NULL;
1046:     temp->direction_xyz= NULL;
1047:     temp->cal_due_date= NULL;
1048:     temp->power_requirements= NULL;
1049:     temp->spa_u_hub= NULL;
1050:     temp->spa_u_port= NULL;
1051:     temp->qualifiers= NULL;
1052:     temp->memory_minimum= NULL;
1053:     temp->operating_system= NULL;
1054:     temp->path_for_assembly= NULL;
1055:     temp->path_on_spacecraft= NULL;
1056:     return temp;
1057: }
1058:
1059: xteds* add_references(xteds* xTEDS,app_dev_attr* attr,interface* face)
1060: {
1061:     xTEDS->header = attr;
1062:     xTEDS->interfaces = face;

```

```

1063: return xTEDS;
1064: }
1065:
1066: int delete_qualifier qualifier_type* p)
1067: {
1068:     if(p==NULL) return 0;
1069:     delete_qualifier(p->next);
1070:     free(p->name);
1071:     free(p->value);
1072:     free(p->units);
1073:     free(p);
1074:     return 1;
1075: }
1076:
1077: int delete_coef(coef* p)
1078: {
1079:     if(p==NULL) return 0;
1080:     delete_coef(p->next);
1081:     free(p->exponent);
1082:     free(p->value);
1083:     free(p->description);
1084:     free(p);
1085:     return 1;
1086: }
1087:
1088: int delete_curve(curve* p)
1089: {
1090:     if(p==NULL) return 0;
1091:     delete_coef(p->coefs);
1092:     free(p->name);
1093:     free(p->description);
1094:     free(p);
1095:     return 1;
1096: }
1097:
1098: int delete_option(curveoption* p)
1099: {
1100:     if(p==NULL) return 0;
1101:     delete_option(p->next);
1102:     free(p->name);
1103:     free(p->value);

```

```

1104: free(p->description);
1105: free(p->alarm);
1106: free(p);
1107: return 1;
1108: }
1109:
1110: int delete_drangle(drangle* p)
1111: {
1112:     if(p==NULL) return 0;
1113:     delete_option(p->options);
1114:     free(p->name);
1115:     free(p->description);
1116:     free(p);
1117:     return 1;
1118: }
1119:
1120: int delete_location(location* p)
1121: {
1122:     if (p==NULL) return 0;
1123:     free(p->x_value);
1124:     free(p->y_value);
1125:     free(p->z_value);
1126:     free(p->units);
1127:     free(p);
1128:     return 1;
1129: }
1130:
1131: int delete_orientation(orientation* p)
1132: {
1133:     if (p==NULL) return 0;
1134:     delete_orientation(p->next);
1135:     free(p->axis_value);
1136:     free(p->angle_value);
1137:     free(p->units_value);
1138:     free(p);
1139:     return 1;
1140: }
1141:
1142: int delete_variable(variable* p)
1143: {
1144:     if(p==NULL) return 0;

```

```

1145: delete_variable(p->next);
1146: delete_qualifier(p->qualifiers);
1147: delete_curve(p->curves);
1148: delete_drangle(p->dranges);
1149: delete_location(p->location_data);
1150: delete_orientation(p->orientation_data);
1151: free(p->length);
1152: free(p->kind);
1153: free(p->name);
1154: free(p->qualifier);
1155: free(p->id);
1156: free(p->range_min);
1157: free(p->range_max);
1158: free(p->default_value);
1159: free(p->precision);
1160: free(p->units);
1161: free(p->accuracy);
1162: free(p->scale_factor);
1163: free(p->scale_units);
1164: free(p->format);
1165: free(p->description);
1166: free(p->interface_name);
1167: free(p->interface_id);
1168: free(p->r_low);
1169: free(p->r_high);
1170: free(p->y_low);
1171: free(p->y_high);
1172: free(p->invalid_value);
1173: free(p);
1174: return 1;
1175: }
1176:
1177: int delete_var_ref(var_ref* p)
1178: {
1179:   if(p==NULL) return 0;
1180:   delete_var_ref(p->next);
1181:   free(p->name);
1182:   free(p);
1183:   return 1;
1184: }
1185:

```

```

1186: int delete_fault_msg(fault_msg* p)
1187: {
1188:     if(p==NULL) return 0;
1189:     delete_var_ref(p->variables);
1190:     delete_qualifier(p->qualifiers);
1191:     free(p->name);
1192:     free(p->id);
1193:     free(p->description);
1194:     free(p->interface_name);
1195:     free(p->interface_id);
1196:     free(p);
1197:     return 1;
1198: }
1199:
1200: int delete_data_msg(data_msg* p)
1201: {
1202:     if(p==NULL) return 0;
1203:     delete_data_msg(p->next);
1204:     delete_var_ref(p->variables);
1205:     delete_qualifier(p->qualifiers);
1206:     free(p->name);
1207:     free(p->id);
1208:     free(p->msg_arrival);
1209:     free(p->description);
1210:     free(p->msg_rate);
1211:     free(p->interface_name);
1212:     free(p->interface_id);
1213:     free(p);
1214:     return 1;
1215: }
1216:
1217: int delete_cmd_msg(cmd_msg* p)
1218: {
1219:     if(p==NULL) return 0;
1220:     delete_cmd_msg(p->next);
1221:     delete_var_ref(p->variables);
1222:     delete_qualifier(p->qualifiers);
1223:     free(p->name);
1224:     free(p->id);
1225:     free(p->description);
1226:     free(p->interface_name);

```



```

1227: free(p->interface_id);
1228: free(p);
1229: return 1;
1230: }
1231:
1232: int delete_command(command* p)
1233: {
1234:     if(p==NULL) return 0;
1235:     delete_command(p->next);
1236:     delete_cmd_msg(p->command_message);
1237:     delete_fault_msg(p->fault_message);
1238:     free(p->interface_name);
1239:     free(p->interface_id);
1240:     free(p);
1241:     return 1;
1242: }
1243:
1244: int delete_notification(notification* p)
1245: {
1246:     if(p==NULL) return 0;
1247:     delete_notification(p->next);
1248:     delete_data_msg(p->data_message);
1249:     delete_fault_msg(p->fault_message);
1250:     free(p->interface_name);
1251:     free(p->interface_id);
1252:     free(p);
1253:     return 1;
1254: }
1255:
1256: int delete_request(request* p)
1257: {
1258:     if(p==NULL) return 0;
1259:     delete_request(p->next);
1260:     delete_cmd_msg(p->command_message);
1261:     delete_data_msg(p->data_message);
1262:     delete_fault_msg(p->fault_message);
1263:     free(p->interface_name);
1264:     free(p->interface_id);
1265:     free(p);
1266:     return 1;
1267: }

```

```

1268:
1269: int delete_message(message* p)
1270: {
1271:     if(p==NULL) return 0;
1272:     delete_command(p->commands);
1273:     delete_notification(p->notifications);
1274:     delete_request(p->requests);
1275:     free(p);
1276:     return 1;
1277: }
1278:
1279: int delete_interface(interface* p)
1280: {
1281:     if(p==NULL) return 0;
1282:     delete_interface(p->next);
1283:     delete_qualifier(p->qualifiers);
1284:     delete_variable(p->variables);
1285:     delete_command(p->commands);
1286:     delete_notification(p->notifications);
1287:     delete_request(p->requests);
1288:     free(p->name);
1289:     free(p->extends);
1290:     free(p->id);
1291:     free(p->description);
1292:     free(p);
1293:     return 1;
1294: }
1295:
1296: extern SDMLIB_API int delete_xteds(xteds* p)
1297: {
1298:     if(p==NULL) return 0;
1299:     delete_app_dev_attr(p->header);
1300:     delete_interface(p->interfaces);
1301:     free(p->name);
1302:     free(p->version);
1303:     free(p->description);
1304:     free(p->xmlns);
1305:     free(p->xmlns_xsi);
1306:     free(p->schema_location);
1307:     free(p);
1308:     return 1;

```

```

1309: }
1310:
1311: int delete_app_dev_attr(app_dev_attr* p)
1312: {
1313:     if(p==NULL) return 0;
1314:     delete_qualifier(p->qualifiers);
1315:     free(p->name);
1316:     free(p->kind);
1317:     free(p->id);
1318:     free(p->qualifier);
1319:     free(p->description);
1320:     free(p->manufacturer_id);
1321:     free(p->componentKey);
1322:     free(p->model_id);
1323:     free(p->version_letter);
1324:     free(p->version);
1325:     free(p->serial_number);
1326:     free(p->calibration_date);
1327:     free(p->sensitivity_at_reference);
1328:     free(p->reference_frequency);
1329:     free(p->reference_temperature);
1330:     free(p->measurement_range);
1331:     free(p->electrical_output);
1332:     free(p->quality_factor);
1333:     free(p->temperature_coefficient);
1334:     free(p->direction_xyz);
1335:     free(p->cal_due_date);
1336:     free(p->power_requirements);
1337:     free(p->spa_u_hub);
1338:     free(p->spa_u_port);
1339:     free(p->memory_minimum);
1340:     free(p->operating_system);
1341:     free(p->path_for_assembly);
1342:     free(p->path_on_spacecraft);
1343:     free(p);
1344:     return 1;
1345: }
1346:
1347: SDMLIB_API variable* getParsedVariable()
1348: {
1349:     return var_result;

```

```
1350: }
1351:
1352: extern SDMLIB_API xteds* parsexTEDS(char* text)
1353: {
1354:     extern void *xTEDS_scan_string(const char* str);
1355:     extern int xTEDSparsed();
1356:     extern void xTEDS_delete_buffer(void*);
1357:     extern int line_count;
1358:
1359:     int parse_result;
1360:     void* buffer;
1361:     line_count = 1;
1362:
1363:     buffer = xTEDS_scan_string(text);
1364:     parse_result=xTEDSparsed();
1365:     xTEDS_delete_buffer(buffer);
1366:     if(parse_result==0)
1367:     {
1368:         return result;
1369:     }
1370:     return NULL;
1371: }
1372:
```

File: sdm/common/xTEDS/xTEDSOptionList.h

```
1: #ifndef __SDM_XTEDS_OPTION_LIST_H_
2: #define __SDM_XTEDS_OPTION_LIST_H_
3: #include <stdlib.h>
4: #include "xTEDSOption.h"
5: extern "C"
6: {
7: #include "xTEDSParser.h"
8: }
9:
10: struct xTEDSOptionListNode
11: {
12: xTEDSOption data;
13: struct xTEDSOptionListNode* next;
14: xTEDSOptionListNode():data(),next(NULL) {}
15: xTEDSOptionListNode(const xTEDSOptionListNode&);
16: xTEDSOptionListNode& operator=(const xTEDSOptionListNode&);
17: };
18:
19: class xTEDSOptionList
20: {
21: public:
22: xTEDSOptionList();
23: xTEDSOptionList(const xTEDSOptionList&);
24: ~xTEDSOptionList();
25: xTEDSOptionList& operator=(const xTEDSOptionList&);
26:
27: void setOptionsList(const curveoption* OptionList);
28: void addOption(const curveoption* Option);
29: bool IsEmpty(void) const;
30:
31: void VarInfoRequest(char* InfoBufferOut, size_t BufferSize) const;
32: private:
33: struct xTEDSOptionListNode* head;
34: struct xTEDSOptionListNode* tail;
35: };
36:
37: #endif
```

File: sdm/common/xTEDS/xTEDSQualifier.h

```
1: #ifndef __SDM_XTEDS_QUALIFIER_H_
2: #define __SDM_XTEDS_QUALIFIER_H_
3:
4: #include "../message_defs.h"
5: #include <cstring>
6:
7: class xTEDSQualifier
8: {
9: public:
10: xTEDSQualifier();
11: xTEDSQualifier(char* pName, char* pDescription, char* pUnits);
12: xTEDSQualifier(const xTEDSQualifier&);
13:
14: virtual ~xTEDSQualifier();
15:
16: void setName(const char*);
17: void setDescription(const char*);
18: void setUnits(const char*);
19: const char* getName(void) const;
20: const char* getDescription(void) const;
21: const char* getUnits(void) const;
22: bool Parse(const char*);
23: void QualifierInfoRequest(char* InfoBufferOut, size_t BufferSize);
24:
25: bool MatchesName(const char* MatchString) const;
26: bool MatchesDescription(const char* MatchString) const;
27:
28: xTEDSQualifier& operator=(const xTEDSQualifier&);
29: private:
30: char* m_strName;
31: char* m_strDescription;
32: char* m_strUnits;
33: };
34:
35: #endif
```

File: sdm/common/xTEDS/xTEDSOrientationItem.h

```
1: #ifndef __SDM_XTEDS_ORIENTATION_ITEM_H_
2: #define __SDM_XTEDS_ORIENTATION_ITEM_H_
3:
4: extern "C"
5: {
6: #include "xTEDSParser.h"
7: }
8:
9: class xTEDSOrientationItem
10: {
11: public:
12: xTEDSOrientationItem();
13: ~xTEDSOrientationItem();
14: xTEDSOrientationItem(const xTEDSOrientationItem&);
15: xTEDSOrientationItem& operator=(const xTEDSOrientationItem&);
16:
17: void SetAxis(const char *);
18: void SetAngle(const char *);
19: void SetUnits(const char *);
20:
21: void SetOrientation(const orientation*);
22:
23: void VarInfoRequest(char* InfoBufferOut, size_t BufferLength) const;
24: private:
25: char *m_strAxis;
26: char *m_strAngle;
27: char *m_strUnits;
28: };
29:
30:
31: #endif
```

File: sdm/common/xTEDS/xTEDSRequest.cpp

```
1: #include "xTEDSRequest.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include <unistd.h>
8: #endif
9:
10:
11: xTEDSRequest::xTEDSRequest():m_ReplyMessage(NULL),m_CommandMessage(NULL),m_FaultMes
sage(NULL)
12: {
13:     Type = WRAPPER_REQUEST;
14: }
15:
16: ~xTEDSRequest()
17: {
18:     if (m_ReplyMessage != NULL)
19:         delete m_ReplyMessage;
20:     if (m_CommandMessage != NULL)
21:         delete m_CommandMessage;
22:     if (m_FaultMessage != NULL)
23:         delete m_FaultMessage;
24: }
25:
26: bool xTEDSRequest::SetRequest(const request* Request, const xTEDSVariableList& VariablesList)
27: {
28:     if (Request == NULL || Request->command_message == NULL || Request->data_message ==
NULL)
29:         return false;
30:
31:     if (Request->interface_name != NULL) setInterfaceName (Request->interface_name);
32:     if (Request->interface_id != NULL) setInterfaceID (atoi(Request->interface_id));
33:
34:     // Set the command message, required
35:     m_CommandMessage = new xTEDSCommandMsg();
36:     bool ReturnResult = m_CommandMessage->SetCommandMsg(Request->command_message,
VariablesList);
37:     if (ReturnResult == false)
```



```

37:     return false;
38:
39: // Set the reply message, required
40: m_ReplyMessage = new xTEDSDataMsg();
41: ReturnResult = m_ReplyMessage->SetDataMsg(Request->data_message, VariablesList);
42: if (ReturnResult == false)
43:     return false;
44:
45: // Set the fault message, optional
46: if (Request->fault_message != NULL)
47: {
48:     m_FaultMessage = new xTEDSFaultMsg();
49:     ReturnResult = m_FaultMessage->SetFaultMsg(Request->fault_message, VariablesList);
50: }
51: return ReturnResult;
52: }
53:
54: MessageDef* xTEDSRequest::RegInfo(const char* ItemName) const
55: {
56: char DefinitionsBuf[MSG_DEF_SIZE] = "";
57: char xTEDSPortionBuf[MSG_DEF_SIZE] = "";
58: char VarPortionBuf[MSG_DEF_SIZE] = "";
59:
60: if (m_ReplyMessage == NULL || m_CommandMessage == NULL) return NULL;
61:
62: MessageDef* pReplyMsgDef;
63: MessageDef* pCmdMsgDef;
64: MessageDef* pFaultMsgDef;
65:
66: //produce data and command message portions
67: pReplyMsgDef = m_ReplyMessage->RegInfo();
68: if (NULL == pReplyMsgDef)
69:     return NULL;
70:
71: VariableDef* pReplyVarDef = m_ReplyMessage->GetVariableXtedsDefinitions();
72:
73: pCmdMsgDef = m_CommandMessage->RegInfo();
74: if (NULL == pCmdMsgDef)
75: {
76:     free (pReplyMsgDef);
77:     free (pReplyVarDef);

```

```

78:     return NULL;
79: }
80: VariableDef* pCmdVarDef = m_CommandMessage->GetVariableXtedsDefinitions();
81:
82: if(m_FaultMessage!=NULL)    // If there exists a fault message
83: {
84:     pFaultMsgDef = m_FaultMessage->RegInfo();
85:     if (pFaultMsgDef == NULL)
86:     {
87:         free (pReplyMsgDef);
88:         free (pReplyVarDef);
89:         free (pCmdMsgDef);
90:         free (pCmdVarDef);
91:         return NULL;
92:     }
93:     VariableDef* pFaultVarDef = m_FaultMessage->GetVariableXtedsDefinitions();
94:
95:     snprintf(DefinitionsBuf, sizeof(DefinitionsBuf), "<Request> \n \t%s \n \t%s \n \t%s
\n</Request>",
96:         pCmdMsgDef->GetDefinitions(),
97:         pReplyMsgDef->GetDefinitions(),
98:         pFaultMsgDef->GetDefinitions());
99:
100:     pCmdMsgDef->Join(pFaultMsgDef);
101:     VariableDef::ToStringConcatVariableDefsIgnoreDuplicates(VarPortionBuf,
102:         sizeof(VarPortionBuf), pCmdVarDef, pReplyVarDef);
103:
104:     snprintf(xTEDSPortionBuf, sizeof(xTEDSPortionBuf),
105:         "%s \n<Request> \n \t%s \n \t%s \n \t%s \n</Request>", VarPortionBuf,
106:         pCmdMsgDef->GetxTEDSPortion(), pReplyMsgDef->GetxTEDSPortion(),
107:         pFaultMsgDef->GetxTEDSPortion());
108:
109:     delete pFaultMsgDef;
110:     pFaultMsgDef = NULL;
111:
112:     delete pFaultVarDef;
113:     pFaultVarDef = NULL;
114: }
115: else    // If this is only the command/data messages
116: {
117:     snprintf(DefinitionsBuf, sizeof(DefinitionsBuf), "<Request> \n \t%s \n \t%s \n</Request>",

```

```

118:         pCmdMsgDef->GetDefinitions(),
119:         pReplyMsgDef->GetDefinitions());
120:
121:     VariableDef::ToStringConcatVariableDefsIgnoreDuplicates(VarPortionBuf,
122:         sizeof(VarPortionBuf), pCmdVarDef, pReplyVarDef);
123:
124:     snprintf(xTEDSPortionBuf, sizeof(xTEDSPortionBuf),
125:         "%s \n<Request> \n \t%s \n \t%s \n</Request>", VarPortionBuf,
126:         pCmdMsgDef->GetxTEDSPortion(), pReplyMsgDef->GetxTEDSPortion());
127: }
128: MessageDef* ReturnMsgDef = new MessageDef();
129:
130: //
131: // Set the message id to the id of the requested item
132: ReturnMsgDef->SetInterfaceMessageID(pCmdMsgDef->GetInterfaceMessageID());
133: if (ItemName != NULL)
134: {
135:     if (m_ReplyMessage->NameEquals(ItemName))
136:         ReturnMsgDef->SetInterfaceMessageID(pReplyMsgDef->GetInterfaceMessageID());
137:     if (m_FaultMessage != NULL && m_FaultMessage->NameEquals(ItemName))
138:         ReturnMsgDef->SetInterfaceMessageID(m_FaultMessage->GetID());
139: }
140:
141: delete pCmdMsgDef;
142: delete pReplyMsgDef;
143: if (NULL != pReplyVarDef)
144:     delete pReplyVarDef;
145: if (NULL != pCmdVarDef)
146:     delete pCmdVarDef;
147:
148: ReturnMsgDef->SetDefinitions(DefinitionsBuf);
149: ReturnMsgDef->SetxTEDSPortion(xTEDSPortionBuf);
150: return ReturnMsgDef;
151: }
152:
153: bool xTEDSRequest::RegInfoMatch(const char* Name, const xTEDSQualifierList& Qualifiers,
const char* Interface) const
154: {
155:     if (m_ReplyMessage != NULL && m_ReplyMessage->RegInfoMatch(Name, Qualifiers,
Interface))
156:         return true;

```

```

157:     else if (m_CommandMessage != NULL && m_CommandMessage->RegInfoMatch(Name,
Qualifiers, Interface))
158:         return true;
159:     else if (m_FaultMessage != NULL && m_FaultMessage->RegInfoMatch(Name, Qualifiers,
Interface))
160:         return true;
161:
162:     return false;
163: }
164:
165: bool xTEDSRequest::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const
char* Interface) const
166: {
167:     if (m_ReplyMessage != NULL && m_ReplyMessage->RegexMatch(Pattern, QualList,
Interface))
168:         return true;
169:     else if (m_CommandMessage != NULL && m_CommandMessage->RegexMatch(Pattern,
QualList, Interface))
170:         return true;
171:     else if (m_FaultMessage != NULL && m_FaultMessage->RegexMatch(Pattern, QualList,
Interface))
172:         return true;
173:
174:     return false;
175: }
176:
177: bool xTEDSRequest::ContainsMessage(const SDMMMessage_ID& MessageID) const
178: {
179:     if (m_ReplyMessage != NULL && m_ReplyMessage->GetID() == MessageID)
180:         return true;
181:     else if (m_CommandMessage != NULL && m_CommandMessage->GetID() == MessageID)
182:         return true;
183:     else if (m_FaultMessage != NULL && m_FaultMessage->GetID() == MessageID)
184:         return true;
185:     return false;
186: }
187:
188: SDMMMessage_ID xTEDSRequest::GetFaultMessageID() const
189: {
190:     if (m_FaultMessage != NULL)
191:         return m_FaultMessage->GetID();
192:     return SDMMMessage_ID('\0', '\0');

```

```

193: }
194:
195: SDMMMessage_ID xTEDSRequest::GetDataMessageID() const
196: {
197:     if (m_ReplyMessage != NULL)
198:         return m_ReplyMessage->GetID();
199:     return SDMMMessage_ID(' \0', ' \0');
200: }
201:
202: #ifndef REMOVE_DEBUG_OUTPUT
203: void xTEDSRequest::PrintDebug() const
204: {
205:     xTEDSWrapper::PrintDebug();
206:     printf(" xTEDSRequest \n");
207:
208:     if (m_CommandMessage!=NULL)
209:         m_CommandMessage->PrintDebug();
210:
211:     if (m_ReplyMessage!=NULL)
212:         m_ReplyMessage->PrintDebug();
213:
214:     if (m_FaultMessage!=NULL)
215:         m_FaultMessage->PrintDebug();
216: }
217: #endif
218:

```

File: sdm/common/xTEDS/xTEDS.h

```
1: #ifndef __XTEDS_H_
2: #define __XTEDS_H_
3:
4: #include "xTEDSItemTree.h"
5: #include "xTEDSItem.h"
6: #include "MessageDef.h"
7: #include "VariableDef.h"
8:
9: #include "../sdmLib.h"
10:
11: class SDMLIB_API xTEDS
12: {
13: public:
14: xTEDS();
15: ~xTEDS();
16: xTEDS(const xTEDS&);
17: xTEDS& operator=(const xTEDS&);
18:
19: bool Parse(char* xteds_text);
20:
21: MessageDef* RegInfo(const char* Name, const char* Qualifier, const char* Device, const char*
Interface);
22: MessageDef* AllRegInfo(const char* Qualifier, const char* Device, const char* Interface);
23: MessageDef* RegexRegInfo(const char* Pattern, const char* Qualifier, const char* Device, const
char* Interface);
24: VariableDef* getVarInfo(const char* strVariableName, const SDMMMessage_ID& idInterface) const;
25:
26: SDMMMessage_ID getServiceDataMsgID(const SDMMMessage_ID& CommandID) const;
27: SDMMMessage_ID getServiceFaultMsgID(const SDMMMessage_ID& CommandID) const;
28: SDMMMessage_ID getCommandFaultMsgID(const SDMMMessage_ID& CommandID) const;
29: SDMMMessage_ID getNotificationFaultMsgID(const SDMMMessage_ID& DataID) const;
30: bool isCommandIdValid(const SDMMMessage_ID& RequestedId) const;
31: bool isServiceIdValid(const SDMMMessage_ID& RequestedId) const;
32:
33: const char* getDeviceName(void) const { return m_strDeviceName; }
34: const char* getComponentKey(void) const { return m_strComponentKey; }
35: const char* getSPAUHub(void) const { return m_strSPAUHub; }
36: const char* getSPAUPort(void) const { return m_strSPAUPort; }
37: private:
```

```
38: xTEDSItemTree m_TreeRoot;
39: char* m_strDeviceName;
40: char* m_strComponentKey;
41: char* m_strSPAUHub;
42: char* m_strSPAUPort;
43: };
44:
45: #endif
```

File: sdm/common/xTEDS/xTEDSCommandMsg.cpp

```
1: #include "xTEDSCommandMsg.h"
2: #include "MessageDef.h"
3: #include "../Exception/SDMBadIndexException.h"
4:
5: #include <string.h>
6: #include <stdlib.h>
7: #include <stdio.h>
8: #ifdef WIN32
9: # include "unistd.h"
10: #endif
11:
12: xTEDSCommandMsg::xTEDSCommandMsg()
13: {
14:     m_ItemType = TYPE_COMMANDMSG;
15: }
16:
17: xTEDSCommandMsg::~~xTEDSCommandMsg()
18: {
19:
20: }
21:
22: bool xTEDSCommandMsg::SetCommandMsg(const command_message* CommandMsg, const
xTEDSVariableList& VariablesList)
23: {
24:     if (CommandMsg == NULL) return false;
25:
26:     // Set all of the attributes of the message
27:     if (CommandMsg->name != NULL) setName(CommandMsg->name);
28:     if (CommandMsg->id != NULL) setMessageID(atoi(CommandMsg->id));
29:     if (CommandMsg->interface_name != NULL) setInterfaceName(CommandMsg->interface_name);
30:     if (CommandMsg->interface_id != NULL) setInterfaceID(atoi(CommandMsg->interface_id));
31:     if (CommandMsg->description != NULL) setDescription(CommandMsg->description);
32:
33:     if (CommandMsg->qualifiers != NULL)
34:     {
35:         for (qualifier_type* CurQual = CommandMsg->qualifiers; CurQual != NULL; CurQual =
CurQual->next)
36:             addQualifier(xTEDSQualifier(CurQual->name, CurQual->value, CurQual->units));
37:     }
```



```

38:
39: // Set the variables
40: for (var_ref* Cur = CommandMsg->variables; Cur != NULL; Cur = Cur->next)
41: {
42:     const xTEDSVariable* CurVar = VariablesList.Find(Cur->name, CommandMsg-
->interface_name);
43:
44:     if (CurVar == NULL)
45:     {
46:         printf("Error, could not find variable %s in interface %s being reference in command
message %s. \n",Cur->name, CommandMsg->interface_name, CommandMsg->name);
47:         return false; // Variable not found, error
48:     }
49:
50:     // Otherwise, add to variables list
51:     m_Variables.addItem(CurVar);
52: }
53: return true;
54: }
55:
56: MessageDef* xTEDSCommandMsg::RegInfo(void) const
57: {
58: char InfoBuf[MSG_DEF_SIZE];
59: char xTEDSPortion[MSG_DEF_SIZE];
60: int BufLength = 0;
61: MessageDef* ReturnDef = new MessageDef();
62:
63: xTEDSPortion[0] = InfoBuf[0] = '\0';
64:
65: snprintf(InfoBuf, sizeof(InfoBuf), "<CommandMsg name= \"%s \" id= \"%d \" ", m_strItemName,
m_ItemInterfaceMessageID.getInterfaceMessagePair());
66: if (!m_Variables.IsEmpty())
67: {
68:     BufLength = strlen(InfoBuf);
69:     m_Variables.VarReqReg(InfoBuf + BufLength, sizeof(InfoBuf) - BufLength);
70: }
71: BufLength = strlen(InfoBuf);
72: strncat(InfoBuf, "/>", sizeof(InfoBuf) - BufLength);
73:
74: ReturnDef->SetInterfaceMessageID(m_ItemInterfaceMessageID);
75: ReturnDef->SetDefinitions(InfoBuf);
76:

```

```

77: // Add xTEDS section
78: xTEDSInfo(xTEDSPortion, sizeof(xTEDSPortion));
79: ReturnDef->SetxTEDSPortion(xTEDSPortion);
80:
81: return ReturnDef;
82: }
83:
84: void xTEDSCommandMsg::xTEDSInfo(char* InfoBufferOut, int BufferSize) const
85: {
86: char xTEDSBuf[MSG_DEF_SIZE];
87: const unsigned int xTEDSBufSize = sizeof(xTEDSBuf);
88: int BufLength = 0;
89:
90: if (m_strItemName == NULL)
91:     return ;
92:
93: snprintf(xTEDSBuf, xTEDSBufSize, "<CommandMsg name= \"%s \" id= \"%d
\"\",m_strItemName,m_ItemInterfaceMessageID.getInterfaceMessagePair());
94: // Get the description
95: if(m_strItemDescription != NULL)
96: {
97:     strncat(xTEDSBuf, " description= \"\", xTEDSBufSize);
98:     strncat(xTEDSBuf, m_strItemDescription, xTEDSBufSize);
99:     strncat(xTEDSBuf, " \"\", xTEDSBufSize);
100: }
101: // Close the tag
102: strncat(xTEDSBuf, ">", xTEDSBufSize);
103: // Add any qualifiers
104: if(m_ItemQualifiers!=NULL)
105: {
106:     BufLength = strlen(xTEDSBuf);
107:     strncat(xTEDSBuf, " \n \t \t", sizeof(xTEDSBuf) - BufLength);
108:
109:     BufLength += 3;
110:     m_ItemQualifiers->QualifierInfoRequest(xTEDSBuf + BufLength, sizeof(xTEDSBuf) -
BufLength);
111: }
112:
113: // Add the variables
114: if (!m_Variables.IsEmpty())
115: {

```

```

116:     BufLength = strlen(xTEDSBuf);
117:     strncat(xTEDSBuf, " \n \t \t", sizeof(xTEDSBuf) - BufLength);
118:
119:     BufLength += 3;
120:     m_Variables.VarRef(xTEDSBuf + BufLength, sizeof(xTEDSBuf) - BufLength);
121: }
122:
123: // Close the CommandMsg
124: strncat(xTEDSBuf, " \n \t</CommandMsg>", xTEDSBufSize);
125:
126: strncat(InfoBufferOut, xTEDSBuf, BufferSize - 1);
127: }
128:
129: bool xTEDSCommandMsg::MatchesQualifier(const xTEDSQualifierList& QualList) const
130: {
131:     if(QualList.isEmpty())
132:         return true;
133:
134:     char id_str[8];
135:     snprintf(id_str, sizeof(id_str), "%d", m_ItemInterfaceMessageID.getMessage());
136:
137:     //check entire qualifier list
138:     for (int i = 0; i < QualList.Size(); i++)
139:     {
140:         try
141:         {
142:             const xTEDSQualifier& CurQual = QualList[i];
143:             if (CurQual.MatchesName("name"))
144:             {
145:                 if (!CurQual.MatchesDescription(m_strItemName))
146:                     return false;
147:             }
148:             else if (CurQual.MatchesName("description"))
149:             {
150:                 if (!CurQual.MatchesDescription(m_strItemDescription))
151:                     return false;
152:             }
153:             else if (CurQual.MatchesName("id"))
154:             {
155:                 if (!CurQual.MatchesDescription(id_str))
156:                     return false;

```

```

157:         }
158:         else
159:             return false;
160:     }
161:     catch (SDMBadIndexException& ex)
162:     {
163:         printf("Error %s \n", ex.Message());
164:         return false;
165:     }
166: }
167: return true;
168: }
169:
170: bool xTEDSCommandMsg::RegInfoMatch(const char* pname, const xTEDSQualifierList&
qualifiers, const char* interface) const
171: {
172:     if (interface != NULL && strcmp(interface, m_strItemInterfaceName) != 0)
173:         return false;
174:
175:     if (pname == NULL || strcmp(pname, m_strItemName)==0)
176:     {
177:         if (MatchesQualifier(qualifiers))
178:         {
179:             return true;
180:         }
181:     }
182:
183:     if (m_Variables.RegInfoMatch(pname, qualifiers, interface))
184:         return true;
185:
186:     return false;
187: }
188:
189: bool xTEDSCommandMsg::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList,
const char* Interface) const
190: {
191:     if (m_Variables.RegexMatch(Pattern, QualList, Interface))
192:         return true;
193:
194:     return xTEDSItem::RegexMatch(Pattern, QualList, Interface);
195: }

```

```
196:
197: #ifndef REMOVE_DEBUG_OUTPUT
198: void xTEDSCommandMsg::PrintDebug() const
199: {
200:     xTEDSMessage::PrintDebug();
201:     printf(" xTEDSCommandMsg \n \n");
202:     m_Variables.PrintDebug();
203: }
204: #endif
205:
```

File: sdm/common/xTEDS/xTEDSNotification.cpp

```
1: #include "xTEDSNotification.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10: xTEDSNotification::xTEDSNotification():m_DataMessage(NULL),m_FaultMessage(NULL)
11: {
12:     Type = WRAPPER_NOTIFICATION;
13: }
14:
15: xTEDSNotification::~~xTEDSNotification()
16: {
17:     if (m_DataMessage != NULL)
18:         delete m_DataMessage;
19:     if (m_FaultMessage != NULL)
20:         delete m_FaultMessage;
21: }
22:
23: bool xTEDSNotification::SetNotification(const notification* Notification, const
xTEDSVariableList& VariablesList)
24: {
25:     if (Notification == NULL || Notification->data_message == NULL)
26:         return false;
27:
28:     if (Notification->interface_name != NULL) setInterfaceName (Notification->interface_name);
29:     if (Notification->interface_id != NULL) setInterfaceID (atoi(Notification->interface_id));
30:
31:     // Set the data message, required
32:     m_DataMessage = new xTEDSDataMsg();
33:     bool ReturnResult = m_DataMessage->SetDataMsg(Notification->data_message, VariablesList);
34:     if (ReturnResult == false)
35:         return false;
36:
37:     // Set the fault message, optional
38:     if (Notification->fault_message != NULL)
```

```

39: {
40:     m_FaultMessage = new xTEDSFaultMsg();
41:     ReturnResult = m_FaultMessage->SetFaultMsg(Notification->fault_message, VariablesList);
42: }
43: return ReturnResult;
44: }
45:
46: MessageDef* xTEDSNotification::RegInfo(const char* ItemName) const
47: {
48:     char DefinitionsBuf[MSG_DEF_SIZE] = "";
49:     char xTEDSPortionBuf[MSG_DEF_SIZE] = "";
50:     char VarPortionBuf[MSG_DEF_SIZE] = "";
51:
52:     if (m_DataMessage == NULL) return NULL;
53:
54:     MessageDef* pDataMsgDef = m_DataMessage->RegInfo();
55:     if (NULL == pDataMsgDef)
56:         return NULL;
57:
58:     VariableDef* pDataVarDef = m_DataMessage->GetVariableXtedsDefinitions();
59:
60:     if(m_FaultMessage!=NULL)    // If there exists a fault message
61:     {
62:         MessageDef* pFaultMsgDef = m_FaultMessage->RegInfo();
63:         if (NULL == pFaultMsgDef)
64:         {
65:             free (pDataMsgDef);
66:             free (pDataVarDef);
67:             return NULL;
68:         }
69:
70:         VariableDef* pFaultVarDef = m_FaultMessage->GetVariableXtedsDefinitions();
71:
72:         snprintf(DefinitionsBuf,    sizeof(DefinitionsBuf),    "<Notification>    \n    \t%s    \n    \t%s
\n</Notification>",
73:             pDataMsgDef->GetDefinitions(),
74:             pFaultMsgDef->GetDefinitions());
75:
76:         VariableDef::ToStringConcatVariableDefsIgnoreDuplicates(VarPortionBuf,
77:             sizeof(VarPortionBuf), pDataVarDef, pFaultVarDef);
78:

```

```

79:     snprintf(xTEDSPortionBuf,sizeof(xTEDSPortionBuf),
80:         "%s \n<Notification> \n \t%s \n \t%s \n</Notification>",
81:         VarPortionBuf,
82:         pDataMsgDef->GetxTEDSPortion(),
83:         pFaultMsgDef->GetxTEDSPortion());
84:
85:     delete pFaultMsgDef;
86:     pFaultMsgDef = NULL;
87:
88:     delete pFaultVarDef;
89:     pFaultVarDef = NULL;
90: }
91: else          // If this is only the data message
92: {
93:     snprintf(DefinitionsBuf,sizeof(DefinitionsBuf),"<Notification> \n \t%s \n</Notification>",
94:         pDataMsgDef->GetDefinitions());
95:
96:     VariableDef::ToStringConcatVariableDefs(VarPortionBuf, sizeof(VarPortionBuf),
97:         pDataVarDef, NULL);
98:
99:     snprintf(xTEDSPortionBuf,sizeof(xTEDSPortionBuf),
100:         "%s \n<Notification> \n \t%s \n</Notification>",
101:         VarPortionBuf, pDataMsgDef->GetxTEDSPortion());
102: }
103:
104: MessageDef* ReturnDef = new MessageDef();
105:
106: //
107: // Set the message id to the id of the requested item
108: ReturnDef->SetInterfaceMessageID(pDataMsgDef->GetInterfaceMessageID());
109: if (ItemName != NULL)
110: {
111:     if (m_FaultMessage != NULL && m_FaultMessage->NameEquals(ItemName))
112:         ReturnDef->SetInterfaceMessageID(m_FaultMessage->GetID());
113: }
114:
115: delete pDataMsgDef;
116:
117: if (NULL != pDataVarDef)
118:     delete pDataVarDef;
119:

```



```

120:   ReturnDef->SetDefinitions(DefinitionsBuf);
121:   ReturnDef->SetxTEDSPortion(xTEDSPortionBuf);
122:
123:   return ReturnDef;
124: }
125:
126: bool xTEDSNotification::RegInfoMatch(const char* Name, const xTEDSQualifierList& Qualifiers,
const char* Interface) const
127: {
128:   if (m_DataMessage != NULL && m_DataMessage->RegInfoMatch(Name, Qualifiers,
Interface))
129:       return true;
130:   else if (m_FaultMessage != NULL && m_FaultMessage->RegInfoMatch(Name, Qualifiers,
Interface))
131:       return true;
132:
133:   return false;
134: }
135:
136: bool xTEDSNotification::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList,
const char* Interface) const
137: {
138:   if (m_DataMessage != NULL && m_DataMessage->RegexMatch(Pattern, QualList, Interface))
139:       return true;
140:   else if (m_FaultMessage != NULL && m_FaultMessage->RegexMatch(Pattern, QualList,
Interface))
141:       return true;
142:
143:   return false;
144: }
145:
146: bool xTEDSNotification::ContainsMessage(const SDMMMessage_ID& MessageID) const
147: {
148:   if (m_DataMessage != NULL && m_DataMessage->GetID() == MessageID)
149:       return true;
150:   else if (m_FaultMessage != NULL && m_FaultMessage->GetID() == MessageID)
151:       return true;
152:   return false;
153: }
154:
155: SDMMMessage_ID xTEDSNotification::GetFaultMessageID() const
156: {

```

```
157:     if (m_FaultMessage != NULL)
158:         return m_FaultMessage->GetID();
159:     return SDMMMessage_ID('\0', '\0');
160: }
161:
162: #ifndef REMOVE_DEBUG_OUTPUT
163: void xTEDSNotification::PrintDebug() const
164: {
165:     xTEDSWrapper::PrintDebug();
166:     printf(" xTEDSNotification \n");
167:
168:     if (m_DataMessage!=NULL)
169:         m_DataMessage->PrintDebug();
170:
171:     if (m_FaultMessage!=NULL)
172:         m_FaultMessage->PrintDebug();
173: }
174: #endif
175:
176:
```

File: sdm/common/xTEDS/xTEDSItem.h

```
1: #ifndef __SDM_XTEDS_ITEM_H_
2: #define __SDM_XTEDS_ITEM_H_
3:
4: #include "MessageDef.h"
5: #include "xTEDSQualifierList.h"
6: #include "../message/SDMMessage_ID.h"
7:
8: class xTEDSItem
9: {
10: public:
11:     enum itemType {TYPE_VARIABLE, TYPE_DATAMSG, TYPE_COMMANDMSG,
12: TYPE_FAULTMSG, TYPE_DATAAREPLYMSG, TYPE_NONE};
13:
14: xTEDSItem();
15: xTEDSItem(const xTEDSItem&);
16:
17: virtual ~xTEDSItem();
18:
19: virtual MessageDef* RegInfo() const = 0;
20: virtual bool RegInfoMatch(const char* name, const xTEDSQualifierList&, const char* interface)
21: const = 0;
22: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
23: Interface) const;
24: void setName(char*);
25: void setDescription(char*);
26: void setInterfaceName(char*);
27: void setInterfaceID(int id);
28: void addQualifier(const xTEDSQualifier &NewQualifier);
29: const char* getName(void) const { return m_strItemName; }
30: const char* getDescription(void) const { return m_strItemDescription; }
31: const char* getInterfaceName(void) const { return m_strItemInterfaceName; }
32: int getInterfaceID() const { return m_ItemInterfaceMessageID.getInterface(); }
33: itemType Type() const { return m_ItemType; }
34: SDMMessage_ID GetID() const { return m_ItemInterfaceMessageID; }
35: bool NameEquals(const char* CompareName) const;
36: xTEDSItem& operator=(const xTEDSItem&);
37: virtual void PrintDebug() const;
38: protected:
39: char* m_strItemName;
40: char* m_strItemDescription;
```

```
38: xTEDSQualifierList* m_ItemQualifiers;
39: virtual bool MatchesQualifier(const xTEDSQualifierList&) const = 0;
40: itemType m_ItemType;
41: char* m_strItemInterfaceName;
42: SDMMMessage_ID m_ItemInterfaceMessageID;
43:
44: };
45:
46: #endif
```

File: sdm/common/xTEDS/xTEDSDataMsg.cpp

```
1: #include "xTEDSDataMsg.h"
2: #include "MessageDef.h"
3: #include "../Exception/SDMBadIndexException.h"
4:
5: #include <string.h>
6: #include <stdlib.h>
7: #include <stdio.h>
8: #ifdef WIN32
9: # include "unistd.h"
10: #endif
11:
12: xTEDSDataMsg::xTEDSDataMsg():msg_arrival(Event),msg_rate(0.0)
13: {
14:     m_ItemType = TYPE_DATAMSG;
15: }
16:
17: xTEDSDataMsg::~~xTEDSDataMsg()
18: {
19:
20: }
21:
22: bool xTEDSDataMsg::SetDataMsg(const data_msg* NewDataMsg, const xTEDSVariableList&
VariablesList)
23: {
24:     if (NewDataMsg == NULL) return false;
25:     // Set the message attributes
26:     if (NewDataMsg->name != NULL) setName(NewDataMsg->name);
27:     if (NewDataMsg->id != NULL) setMessageID(atoi(NewDataMsg->id));
28:     if (NewDataMsg->interface_name != NULL) setInterfaceName(NewDataMsg->interface_name);
29:     if (NewDataMsg->interface_id != NULL) setInterfaceID(atoi(NewDataMsg->interface_id));
30:     if (NewDataMsg->msg_rate != NULL) msg_rate = atof(NewDataMsg->msg_rate);
31:     if (NewDataMsg->description != NULL) setDescription(NewDataMsg->description);
32:
33:     if (NewDataMsg->msg_arrival != NULL)
34:     {
35:         if (strcmp(NewDataMsg->msg_arrival, "PERIODIC")==0)
36:             msg_arrival = Periodic;
37:         else
38:             msg_arrival = Event;
```

```

39: }
40: if (NewDataMsg->qualifiers != NULL)
41: {
42:     for (qualifier_type* CurQual = NewDataMsg->qualifiers; CurQual != NULL; CurQual =
CurQual->next)
43:         addQualifier (xTEDSQualifier(CurQual->name, CurQual->value, CurQual->units));
44: }
45:
46: // Add variables
47: for (var_ref* Cur = NewDataMsg->variables; Cur != NULL; Cur = Cur->next)
48: {
49:     const xTEDSVariable* CurVar = VariablesList.Find(Cur->name, NewDataMsg-
>interface_name);
50:
51:     if (CurVar == NULL)
52:     {
53:         printf("Error, could not find variable %s in interface %s being reference in data message %s.
\n",Cur->name, NewDataMsg->interface_name, NewDataMsg->name);
54:         return false;// Variable not found, error
55:     }
56:
57:     // Otherwise, add to variables list
58:     m_Variables.addItem(CurVar);
59: }
60: return true;
61: }
62:
63: /* Get a RegInfo request about this message */
64: MessageDef* xTEDSDataMsg::RegInfo(void) const
65: {
66:     char InfoBuf[MSG_DEF_SIZE];
67:     char xTEDSPortion[MSG_DEF_SIZE];
68:     int BufLength = 0;
69:     MessageDef* ReturnDef = new MessageDef();
70:
71:     if (m_strItemName == NULL)
72:         return ReturnDef;
73:
74:     xTEDSPortion[0] = InfoBuf[0] = '\0';
75:
76:     snprintf(InfoBuf, sizeof(InfoBuf), "<DataMsg name= \"%s \" id= \"%d \" ", m_strItemName,
m_ItemInterfaceMessageID.getInterfaceMessagePair());

```

```

77: if (!m_Variables.IsEmpty())
78: {
79:     BufLength = strlen(InfoBuf);
80:     m_Variables.VarReqReg(InfoBuf + BufLength, sizeof(InfoBuf) - BufLength);
81: }
82: BufLength = strlen(InfoBuf);
83: strncat(InfoBuf, ">", sizeof(InfoBuf) - BufLength);
84:
85: ReturnDef->SetInterfaceMessageID(m_ItemInterfaceMessageID);
86: ReturnDef->SetDefinitions(InfoBuf);
87:
88: // Add xTEDS section
89: xTEDSInfo(xTEDSPortion, sizeof(xTEDSPortion));
90: ReturnDef->SetxTEDSPortion(xTEDSPortion);
91:
92: return ReturnDef;
93: }
94:
95: /* Create the xTEDS section corresponding to this Data message */
96: void xTEDSDataMsg::xTEDSInfo(char* InfoBufferOut, int BufferSize) const
97: {
98: char xTEDSBuf[MSG_DEF_SIZE];
99: const unsigned int xTEDSBufSize = sizeof(xTEDSBuf);
100:     int BufLength = 0;
101:
102:     if (m_strItemName == NULL)
103:         return ;
104:     // Start with the empty string
105:     xTEDSBuf[0] = '\0';
106:
107:     switch(msg_arrival)
108:     {
109:         case Event:
110:             snprintf(xTEDSBuf, xTEDSBufSize, "<DataMsg name= \"%s\" id= \"%d\" \"",
msgArrival= "EVENT \"",m_strItemName,m_ItemInterfaceMessageID.getInterfaceMessagePair());
111:             break;
112:         case Periodic:
113:             if(msg_rate != 0.0f)
114:                 snprintf(xTEDSBuf, xTEDSBufSize, "<DataMsg name= \"%s\" id= \"%d\" \"",
msgArrival= "\"PERIODIC\" \" msgRate= \"%f\" \"", m_strItemName,
m_ItemInterfaceMessageID.getInterfaceMessagePair(), msg_rate);
115:             else

```

```

116:         snprintf(xTEDSBuf, xTEDSBufSize, "<DataMsg name= \"%s \" id= \"%d \"
msgArrival= \"PERIODIC \"",m_strItemName, m_ItemInterfaceMessageID.getInterfaceMessagePair());
117:         break;
118:         default:
119:             printf("Warning -- xTEDSDataMsg::xTEDSInfo - msg_arrival is invalid. \n");
120:     }
121:     // Add the description
122:     if(m_strItemDescription != NULL)
123:     {
124:         strncat(xTEDSBuf, " description= \"", xTEDSBufSize);
125:         strncat(xTEDSBuf, m_strItemDescription, xTEDSBufSize);
126:         strncat(xTEDSBuf, "\"", xTEDSBufSize);
127:     }
128:     // Close the tag
129:     strncat(xTEDSBuf, ">", xTEDSBufSize);
130:     // Add any qualifier elements
131:     if(m_ItemQualifiers!=NULL)
132:     {
133:         BufLength = strlen(xTEDSBuf);
134:         strncat(xTEDSBuf, " \n \t \t", sizeof(xTEDSBuf) - BufLength);
135:
136:         BufLength += 3;
137:         m_ItemQualifiers->QualifierInfoRequest(xTEDSBuf + BufLength, sizeof(xTEDSBuf) -
BufLength);
138:     }
139:     // Add any variables
140:     if (!m_Variables.IsEmpty())
141:     {
142:         BufLength = strlen(xTEDSBuf);
143:         strncat(xTEDSBuf, " \n \t \t", sizeof(xTEDSBuf) - BufLength);
144:
145:         BufLength += 3;
146:         m_Variables.VarRef(xTEDSBuf + BufLength, sizeof(xTEDSBuf) - BufLength);
147:     }
148:     // Close the CommandMsg
149:     strncat(xTEDSBuf, " \n \t</DataMsg>", xTEDSBufSize);
150:
151:     strncat(InfoBufferOut, xTEDSBuf, BufferSize - 1);
152: }
153:
154: bool xTEDSDataMsg::MatchesQualifier(const xTEDSQualifierList& QualList) const

```



```

155: {
156:     if(QualList.isEmpty())
157:         return true;
158:
159:     char msg_rate_str[16], id_str[8];
160:     snprintf(msg_rate_str, sizeof(msg_rate_str), "%f", msg_rate);
161:     snprintf(id_str, sizeof(id_str), "%d", m_ItemInterfaceMessageID.getMessage());
162:
163:     //check entire qualifier list
164:     for (int i = 0; i < QualList.Size(); i++)
165:     {
166:         try
167:         {
168:             const xTEDSQualifier& CurQual = QualList[i];
169:             if (CurQual.MatchesName("name"))
170:             {
171:                 if (!CurQual.MatchesDescription(m_strItemName))
172:                     return false;
173:             }
174:             else if (CurQual.MatchesName("description"))
175:             {
176:                 if (!CurQual.MatchesDescription(m_strItemDescription))
177:                     return false;
178:             }
179:             else if (CurQual.MatchesName("id"))
180:             {
181:                 if (!CurQual.MatchesDescription(id_str))
182:                     return false;
183:             }
184:             else if (CurQual.MatchesName("msgArrival"))
185:             {
186:                 if (!CurQual.MatchesDescription((msg_arrival==Event?"EVENT":"PERIODIC")))
187:                     return false;
188:             }
189:             else if (CurQual.MatchesName("msgRate"))
190:             {
191:                 if (!CurQual.MatchesDescription(msg_rate_str))
192:                     return false;
193:             }
194:             else
195:                 return false;

```

```

196:     }
197:     catch (SDMBadIndexException& ex)
198:     {
199:         printf("Error %s \n", ex.Message());
200:         return false;
201:     }
202: }
203: return true;
204: }
205:
206: bool xTEDSDataMsg::RegInfoMatch(const char* pname, const xTEDSQualifierList& qualifiers,
const char* interface) const
207: {
208:     if (interface != NULL && strcmp(interface, m_strItemInterfaceName) != 0)
209:         return false;
210:
211:     if (pname == NULL || strcmp(pname, m_strItemName) == 0)
212:     {
213:         if (MatchesQualifier(qualifiers))
214:             return true;
215:     }
216:
217:     if (m_Variables.RegInfoMatch(pname, qualifiers, interface))
218:         return true;
219:
220:     return false;
221: }
222:
223: bool xTEDSDataMsg::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList,
const char* Interface) const
224: {
225:     if (m_Variables.RegexMatch(Pattern, QualList, Interface))
226:         return true;
227:
228:     return xTEDSItem::RegexMatch(Pattern, QualList, Interface);
229: }
230:
231: #ifndef REMOVE_DEBUG_OUTPUT
232: void xTEDSDataMsg::PrintDebug() const
233: {
234:     char MsgArrivalStr[128];

```

```
235:  switch(msg_arrival)
236:  {
237:      case Event:
238:          strcpy(MsgArrivalStr, "Event");
239:          break;
240:      case Periodic:
241:          strcpy(MsgArrivalStr, "Periodic");
242:          break;
243:  default:
244:      break;
245:  }
246:  xTEDSMessage::PrintDebug();
247:  printf("  xTEDSDataMsg msg_arrival %s msg_rate %f \n \n", MsgArrivalStr, msg_rate);
248:  m_Variables.PrintDebug();
249: }
250: #endif
251:
252:
253:
```

File: sdm/common/xTEDS/xTEDSFaultMsg.cpp

```
1: #include "xTEDSFaultMsg.h"
2: #include "MessageDef.h"
3: #include "../Exception/SDMBadIndexException.h"
4: #include <string.h>
5: #include <stdlib.h>
6: #include <stdio.h>
7: #ifdef WIN32
8: # include "unistd.h"
9: #endif
10:
11: xTEDSFaultMsg::xTEDSFaultMsg()
12: {
13: m_ItemType = TYPE_FAULTMSG;
14: }
15:
16: xTEDSFaultMsg::~~xTEDSFaultMsg()
17: {
18:
19: }
20:
21: bool xTEDSFaultMsg::SetFaultMsg(const fault_msg* FaultMsg, const xTEDSVariableList&
VariablesList)
22: {
23: if (FaultMsg == NULL) return false;
24:
25: if (FaultMsg->id != NULL) setMessageID(atoi(FaultMsg->id));
26: if (FaultMsg->name != NULL) setName(FaultMsg->name);
27: if (FaultMsg->description != NULL) setDescription(FaultMsg->description);
28: if (FaultMsg->interface_name != NULL) setInterfaceName(FaultMsg->interface_name);
29: if (FaultMsg->interface_id != NULL) setInterfaceID(atoi(FaultMsg->interface_id));
30: if (FaultMsg->qualifiers != NULL)
31: {
32:     for (qualifier_type* CurQual = FaultMsg->qualifiers; CurQual != NULL; CurQual = CurQual-
>next)
33:         addQualifier (xTEDSQualifier(CurQual->name, CurQual->value, CurQual->units));
34: }
35:
36: // Add variables
37: for (var_ref* Cur = FaultMsg->variables; Cur != NULL; Cur = Cur->next)
```

```

38: {
39:     const xTEDSVariable* CurVar = VariablesList.Find(Cur->name, FaultMsg->interface_name);
40:
41:     if (CurVar == NULL)
42:     {
43:         printf("Error, could not find variable %s in interface %s being reference in fault message %s.
\n",Cur->name, FaultMsg->interface_name, FaultMsg->name);
44:         return false;// Variable not found, error
45:     }
46:
47:     // Otherwise, add to variables list
48:     m_Variables.addItem(CurVar);
49: }
50: return true;
51: }
52:
53: MessageDef* xTEDSFaultMsg::RegInfo(void) const
54: {
55: char InfoBuf[MSG_DEF_SIZE];
56: char xTEDSPortion[MSG_DEF_SIZE];
57: int BufLength = 0;
58: MessageDef* ReturnDef = new MessageDef();
59:
60: xTEDSPortion[0] = InfoBuf[0] = '\0';
61:
62: snprintf(InfoBuf, sizeof(InfoBuf), "<FaultMsg name= \"%s \" id= \"%d \" ", m_strItemName,
m_ItemInterfaceMessageID.getInterfaceMessagePair());
63: if (!m_Variables.IsEmpty())
64: {
65:     BufLength = strlen(InfoBuf);
66:     m_Variables.VarReqReg(InfoBuf + BufLength, sizeof(InfoBuf) - BufLength);
67: }
68: BufLength = strlen(InfoBuf);
69: strncat(InfoBuf, "/>", sizeof(InfoBuf) - BufLength);
70:
71: ReturnDef->SetInterfaceMessageID(m_ItemInterfaceMessageID);
72: ReturnDef->SetDefinitions(InfoBuf);
73:
74: // Add xTEDS section
75: xTEDSInfo(xTEDSPortion, sizeof(xTEDSPortion));
76: ReturnDef->SetxTEDSPortion(xTEDSPortion);

```

```

77:
78: return ReturnDef;
79: }
80:
81: /* Create the xTEDS section corresponding to this fault message */
82: void xTEDSFaultMsg::xTEDSInfo(char* InfoBufferOut, int BufferSize) const
83: {
84: char xTEDSBuf[MSG_DEF_SIZE];
85: const unsigned int xTEDSBufSize = sizeof(xTEDSBuf);
86: int BufLength = 0;
87:
88: if (m_strItemName == NULL)
89:     return ;
90:
91: snprintf(xTEDSBuf, xTEDSBufSize, "<FaultMsg name= \"%s\" id= \"%d\"\"",
m_strItemName, m_ItemInterfaceMessageID.getInterfaceMessagePair());
92: // Add the description
93: if (m_strItemDescription != NULL)
94: {
95:     strcat(xTEDSBuf, " description= \"", xTEDSBufSize);
96:     strcat(xTEDSBuf, m_strItemDescription, xTEDSBufSize);
97:     strcat(xTEDSBuf, "\"", xTEDSBufSize);
98: }
99: // Close the tag
100: strcat(xTEDSBuf, ">", xTEDSBufSize);
101: // Add any qualifiers
102: if (m_ItemQualifiers != NULL)
103: {
104:     BufLength = strlen(xTEDSBuf);
105:     strcat(xTEDSBuf, " \n \t \t", sizeof(xTEDSBuf) - BufLength);
106:
107:     BufLength += 3;
108:     m_ItemQualifiers->QualifierInfoRequest(xTEDSBuf + BufLength, sizeof(xTEDSBuf) -
BufLength);
109: }
110: // Add any variables
111: if (!m_Variables.IsEmpty())
112: {
113:     BufLength = strlen(xTEDSBuf);
114:     strcat(xTEDSBuf, " \n \t \t", sizeof(xTEDSBuf) - BufLength);
115:

```

```

116:     BufLength += 3;
117:     m_Variables.VarRef(xTEDSBuf + BufLength, sizeof(xTEDSBuf) - BufLength);
118: }
119: // Close the FaultMsg
120: strncat(xTEDSBuf, " \n \t</FaultMsg>", xTEDSBufSize);
121:
122: strncat(InfoBufferOut, xTEDSBuf, BufferSize - 1);
123: }
124:
125: bool xTEDSFaultMsg::MatchesQualifier(const xTEDSQualifierList& QualList) const
126: {
127:     if(QualList.isEmpty())
128:         return true;
129:
130:     char id_str[8];
131:     snprintf(id_str, sizeof(id_str), "%d", getMessageID());
132:
133:     //check entire qualifier list
134:     for (int i = 0; i < QualList.Size(); i++)
135:     {
136:         try
137:         {
138:             const xTEDSQualifier& CurQual = QualList[i];
139:             if (CurQual.MatchesName("name"))
140:             {
141:                 if (!CurQual.MatchesDescription(m_strItemName))
142:                     return false;
143:             }
144:             else if (CurQual.MatchesName("description"))
145:             {
146:                 if (!CurQual.MatchesDescription(m_strItemDescription))
147:                     return false;
148:             }
149:             else if (CurQual.MatchesName("id"))
150:             {
151:                 if (!CurQual.MatchesDescription(id_str))
152:                     return false;
153:             }
154:             else
155:                 return false;
156:         }

```

```

157:     catch (SDMBadIndexException& ex)
158:     {
159:         printf("Error %s \n", ex.Message());
160:         return false;
161:     }
162: }
163: return true;
164: }
165:
166: bool xTEDSFaultMsg::RegInfoMatch(const char* pname, const xTEDSQualifierList& qualifiers,
const char* interface) const
167: {
168:     if (pname == NULL || strcmp(pname, m_strItemName)==0)
169:     {
170:         if (interface == NULL || strcmp(interface, m_strItemInterfaceName)==0)
171:         {
172:             if (MatchesQualifier(qualifiers))
173:                 return true;
174:         }
175:     }
176:
177:     if (m_Variables.RegInfoMatch(pname, qualifiers, interface))
178:         return true;
179:
180:     return false;
181: }
182:
183: bool xTEDSFaultMsg::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList,
const char* Interface) const
184: {
185:     if (m_Variables.RegexMatch(Pattern, QualList, Interface))
186:         return true;
187:
188:     return xTEDSItem::RegexMatch(Pattern, QualList, Interface);
189: }
190:
191: #ifndef REMOVE_DEBUG_OUTPUT
192: void xTEDSFaultMsg::PrintDebug() const
193: {
194:     xTEDSMessage::PrintDebug();
195:     printf("  xTEDSFaultMsg \n \n");

```



```
196:    m_Variables.PrintDebug();
197: }
198: #endif
199:
200:
201:
```

File: sdm/common/xTEDS/xTEDSDrange.h

```
1: #ifndef __SDM_XTEDS_DRANGE_H_
2: #define __SDM_XTEDS_DRANGE_H_
3:
4: #include "../message_defs.h"
5: #include "xTEDSOptionList.h"
6: extern "C"
7: {
8: #include "xTEDSParser.h"
9: }
10:
11: class xTEDSDrange
12: {
13: public:
14: xTEDSDrange();
15: xTEDSDrange(const xTEDSDrange&);
16: xTEDSDrange& operator=(const xTEDSDrange&);
17: ~xTEDSDrange();
18:
19: void setName(const char*);
20: void setDescription(const char*);
21: void setDRange(const drange* NewRange);
22:
23: void VarInfoRequest(char* InfoBufferOut, size_t BufferSize) const;
24: private:
25: char* m_strName;
26: char* m_strDescription;
27: xTEDSOptionList m_xolOptions;
28: };
29:
30: #endif
```

File: sdm/common/xTEDS/Makefile

```
1: include ../../Makefile.common
2: include ../../$(MAKEFILE_DEFS)
3:
4: .PHONY:    all clean distclean
5:
6: all: lex.xTEDS.o  xTEDS.tab.o  xTEDSParser.o  xTEDS.o  xTEDSItemTree.o  xTEDSItem.o
xTEDSCommand.o  xTEDSDataMsg.o  xTEDSVariableList.o  xTEDSMessage.o  xTEDSVariable.o
MessageDef.o  xTEDSQualifier.o  xTEDSQualifierList.o  xTEDSNotification.o  xTEDSCommandMsg.o
xTEDSRequest.o  xTEDSFaultMsg.o  xTEDSDrange.o  xTEDSOption.o  xTEDSOptionList.o
xTEDSCoef.o  xTEDSCoefList.o  xTEDSCurve.o  xTEDSLocation.o  xTEDSOrientationList.o
xTEDSOrientationItem.o  VariableDef.o  xTEDSWrapper.o  xTEDSWrapperList.o  xTEDSVerification.o
7:
8: test: test2.o
9: $(CXX) $(CXXFLAGS) -o $@ $^ -lpthread -lboost_regex -L../ -lSDM
10:
11: xTEDSParser.o: xTEDSParser.c xTEDSParser.h
12: $(CC) $(CFLAGS) -fPIC -c $<
13:
14: lex.yy.o:    lex.xTEDS.c xTEDS.tab.c
15: $(CC) $(CFLAGS) -fPIC -c $<
16:
17: lex.xTEDS.o: lex.xTEDS.c
18: $(CC)$(CFLAGS) -fPIC -c $<
19:
20: xTEDS.tab.o:    xTEDS.tab.c
21: $(CC) $(CFLAGS) -fPIC -c $<
22:
23: lex.xTEDS.c:    xTEDS.l xTEDS.tab.c
24: $(LEX) $(LEXFLAGS) -PxTEDS $<
25:
26: xTEDS.tab.c:    xTEDS.y xTEDSParser.h
27: $(YACC) $(YACCFLAGS) -p xTEDS $<
28:
29: xTEDS.o: xTEDS.cpp xTEDSParser.h xTEDS.h
30: $(CXX) $(CXXFLAGS) -fPIC -c $<
31:
32: %.o:    %.cpp %.h
33: $(CXX) $(CXXFLAGS) -fPIC -c $<
34:
35:
```

```
36: clean:
37: rm -f *.o *~
38:
39: distclean: clean
40: rm -f test msgdef.output xTEDS.output
```

File: sdm/common/xTEDS/xTEDSMessage.h

```
1: #ifndef __SDM_XTEDS_MESSAGE_H_
2: #define __SDM_XTEDS_MESSAGE_H_
3:
4: #include "xTEDSItem.h"
5: #include "xTEDSVariableList.h"
6: #include "xTEDSVariable.h"
7:
8: class xTEDSMessage:public xTEDSItem
9: {
10: public:
11: xTEDSMessage();
12: xTEDSMessage(const xTEDSMessage&);
13: virtual ~xTEDSMessage();
14:
15: xTEDSMessage& operator=(const xTEDSMessage&);
16:
17: bool addVariable(xTEDSVariable* );
18:
19: int getMessageID() const;
20: void setMessageID(int NewID);
21: virtual VariableDef* GetVariableXtedsDefinitions() const;
22: virtual void PrintDebug() const;
23: protected:
24: xTEDSVariableList m_Variables;
25: };
26:
27: #endif
```

File: sdm/common/xTEDS/xTEDSOption.h

```
1: #ifndef __SDM_XTEDS_OPTION_H_
2: #define __SDM_XTEDS_OPTION_H_
3:
4: #include "../message_defs.h"
5: #include <cstring>
6: extern "C"
7: {
8: #include "xTEDSParser.h"
9: }
10:
11: class xTEDSOption
12: {
13: public:
14: xTEDSOption();
15: xTEDSOption(const xTEDSOption&);
16:
17: ~xTEDSOption();
18:
19: void setName(const char*);
20: void setDescription(const char*);
21: void setValue(const char*);
22: void setAlarm(const char*);
23: void setOption(const curveoption* Option);
24:
25: void VarInfoRequest(char* InfoBufferOut, size_t BufferSize) const;
26:
27: xTEDSOption& operator=(const xTEDSOption&);
28: private:
29: char* m_strName;
30: char* m_strDescription;
31: char* m_strValue;
32: char* m_strAlarm;
33: };
34:
35: #endif
```

File: sdm/common/xTEDS/xTEDS.cpp

```
1: #include <stdlib.h>
2: #include <string.h>
3:
4: #include "xTEDS.h"
5: extern "C"
6: {
7: #include "xTEDSParser.h"
8: }
9: #include "xTEDSVariable.h"
10: #include "xTEDSDataMsg.h"
11: #include "xTEDSCommandMsg.h"
12: #include "xTEDSQualifierList.h"
13: #include "MessageDef.h"
14: #include "xTEDSRequest.h"
15: #include "xTEDSNotification.h"
16: #include "xTEDSCommand.h"
17: #include "xTEDSVerification.h"
18:
19: // #define DEBUG_OUTPUT_XTEDS_TREE    1
20:
21:
xTEDS::xTEDS():m_TreeRoot(),m_strDeviceName(NULL),m_strComponentKey(NULL),m_strSPAUHub(NULL),m_strSPAUPort(NULL)
22: {}
23:
24:
xTEDS::xTEDS(const xTEDS&
b):m_TreeRoot(b.m_TreeRoot),m_strDeviceName(NULL),m_strComponentKey(NULL),m_strSPAUHub(b(NULL),m_strSPAUPort(NULL)
25: {
26: m_strDeviceName = strdup(b.m_strDeviceName);
27: m_strComponentKey = strdup(b.m_strComponentKey);
28: m_strSPAUHub = strdup(b.m_strSPAUHub);
29: m_strSPAUPort = strdup(b.m_strSPAUPort);
30: }
31:
32: xTEDS::~~xTEDS()
33: {
34: if(m_strDeviceName!=NULL)
35:     free(m_strDeviceName);
36: if(m_strComponentKey!=NULL)
```

```

37:     free(m_strComponentKey);
38: if(m_strSPAUHub!=NULL)
39:     free(m_strSPAUHub);
40: if(m_strSPAUPort!=NULL)
41:     free(m_strSPAUPort);
42: }
43:
44: xTEDS& xTEDS::operator=(const xTEDS&b)
45: {
46: m_TreeRoot = b.m_TreeRoot;
47: m_strDeviceName = strdup(b.m_strDeviceName);
48: m_strComponentKey = strdup(b.m_strComponentKey);
49: m_strSPAUHub = strdup(b.m_strSPAUHub);
50: m_strSPAUPort = strdup(b.m_strSPAUPort);
51: return *this;
52: }
53:
54: bool addVariable(xTEDSItemTree* root,variable* var)
55: {
56: if (!xTEDSVerification::VerifyVariable(var))
57:     return false;
58: // Otherwise, everything is valid
59: root->AddVariable(var);
60: return true;
61: }
62:
63: bool addCommand(xTEDSItemTree* root,const command* cmd)
64: {
65: if (cmd == NULL || root == NULL)
66:     return false;
67: // Add the required command message
68: if (!xTEDSVerification::VerifyCommandMsg(cmd->command_message))
69: {
70:     printf("Invalid command message in <Command> in interface %s \n",cmd->interface_name);
71:     return false;
72: }
73:
74: // Add the optional fault message
75: if(cmd->fault_message != NULL)
76: {
77:     if (!xTEDSVerification::VerifyFaultMsg(cmd->fault_message))

```



```

78:     {
79:         printf("Invalid fault message in <Command> in interface %s \n",cmd->interface_name);
80:         return false;
81:     }
82: }
83:
84: return root->AddCommand(cmd);
85: }
86:
87: bool addRequest(xTEDSItemTree* root,request* svc)
88: {
89: // Add the required data message
90: if (!xTEDSVerification::VerifyDataMsg(svc->data_message))
91: {
92:     printf("Invalid data message in <Request> in interface %s \n",svc->interface_name);
93:     return false;
94: }
95:
96: if (!xTEDSVerification::VerifyCommandMsg(svc->command_message))
97: {
98:     printf("Invalid command message in <Request> in interface %s \n",svc->interface_name);
99:     return false;
100: }
101:
102: // Add the optional fault message
103: if(svc->fault_message != NULL)
104: {
105:     if (!xTEDSVerification::VerifyFaultMsg(svc->fault_message))
106:     {
107:         printf("Invalid fault message in <Request> in interface %s \n",svc->interface_name);
108:         return false;
109:     }
110: }
111:
112: return root->AddRequest(svc);
113: }
114:
115: bool addNotification(xTEDSItemTree* root,notification* noti)
116: {
117: // Add the required data message
118: if (!xTEDSVerification::VerifyDataMsg(noti->data_message))

```

```

119:  {
120:      printf("Invalid data message in <Notification> in interface %s \n",noti->interface_name);
121:      return false;
122:  }
123:
124:  // Add the optional fault message
125:  if(noti->fault_message != NULL)
126:  {
127:      if (!xTEDSVerification::VerifyFaultMsg(noti->fault_message))
128:      {
129:          printf("Invalid fault message in <Notification> in interface %s \n",noti->interface_name);
130:          return false;
131:      }
132:
133:  }
134:  return root->AddNotification(noti);
135: }
136:
137: bool addInterface(xTEDSItemTree* root, interface* face)
138: {
139:     bool result = true;
140:     if(face->name == NULL)
141:     {
142:         printf("Error no name field in the interface tag! \n");
143:         return false;
144:     }
145:     //add variables
146:     for(variable* cur_var=face->variables;cur_var!=NULL;cur_var=cur_var->next)
147:     {
148:         result = addVariable(root,cur_var);
149:         if(result == false)
150:             return false;
151:     }
152:     //add data messages
153:     for(command* cur_cmd=face->commands;cur_cmd!=NULL;cur_cmd=cur_cmd->next)
154:     {
155:         result = addCommand(root,cur_cmd);
156:         if(result == false)
157:             return false;
158:     }
159:     //add command messages

```

```

160:   for(notification* cur_not=face->notifications;cur_not!=NULL;cur_not=cur_not->next)
161:   {
162:       result = addNotification(root,cur_not);
163:       if(result == false)
164:           return false;
165:   }
166:   //add requests
167:   for(request* cur_request=face->requests;cur_request!=NULL;cur_request=cur_request->next)
168:   {
169:       result = addRequest(root,cur_request);
170:       if(result == false)
171:           return false;
172:   }
173:   return true;
174: }
175:
176: bool xTEDS::Parse(char* xteds_text)
177: {
178:     bool result = true;
179:     xteds* parsedXtedsTree = parsexTEDS(xteds_text);
180:     if(parsedXtedsTree == NULL) return false; //syntax error in xTEDS
181:     if(parsedXtedsTree->name == NULL)
182:     {
183:         printf("Error xTEDS tag is missing the name field that is required! \n");
184:         delete_xteds(parsedXtedsTree);
185:         return false;
186:     }
187:     //fill in member data
188:     //TODO add other header fields of importance
189:     if(parsedXtedsTree->header->name!=NULL)
190:     {
191:         if(m_strDeviceName!=NULL) free(m_strDeviceName); //to prevent a memory leak if Parse
is called twice
192:         m_strDeviceName = strdup(parsedXtedsTree->header->name);
193:         if(parsedXtedsTree->header->componentKey!=NULL)
194:             m_strComponentKey = strdup(parsedXtedsTree->header->componentKey);
195:         if(parsedXtedsTree->header->spa_u_hub!=NULL)
196:             m_strSPAUHub = strdup(parsedXtedsTree->header->spa_u_hub);
197:         if(parsedXtedsTree->header->spa_u_port!=NULL)
198:             m_strSPAUPort = strdup(parsedXtedsTree->header->spa_u_port);
199:     }

```

```

200: else
201: {
202:     printf("Error Device or Application tag is missing the name field! \n");
203:     result = false;
204: }
205: if(parsedXtedsTree->header->kind == NULL)
206: {
207:     printf("Error Device or Application tag is missing the kind field! \n");
208:     result = false;
209: }
210: //add interfaces
211: if(result == true)
212: {
213:     for(interface* cur_interface=parsedXtedsTree-
>interfaces;cur_interface!=NULL;cur_interface=cur_interface->next)
214:     {
215:         if (!xTEDSVerification::VerifyInterface(cur_interface))
216:         {
217:             result = false;
218:             break;
219:         }
220:         result = addInterface(&m_TreeRoot,cur_interface);
221:         if(result == false)
222:             break;
223:     }
224:     if(parsedXtedsTree->interfaces == NULL)
225:     {
226:         printf("Error: At least one interface is required! \n");
227:         result = false;
228:     }
229: }
230: //delete temporary structure
231: delete_xteds(parsedXtedsTree);
232: #if (defined (DEBUG_OUTPUT_XTEDS_TREE) && !defined (REMOVE_DEBUG_OUTPUT))
233: if (result == true)
234: {
235:     printf("***** Device %s *****
\n",m_strDeviceName);
236:     m_TreeRoot.PrintDebug();
237:     printf("***** \n");
238:     fflush(NULL);

```

```

239:     }
240: #endif
241:     return result;
242: }
243:
244: MessageDef* xTEDS::RegInfo(const char* ItemName, const char* Qualifiers, const char* Device,
const char* Interface)
245: {
246:     xTEDSQualifierList QualList;
247:     if(Device==NULL || strcmp(Device,m_strDeviceName)==0)
248:     {
249:         QualList.Parse(Qualifiers);
250:         return m_TreeRoot.ExactRegInfo(ItemName, QualList, Interface);
251:     }
252:     else
253:         return NULL;
254: }
255:
256: MessageDef* xTEDS::AllRegInfo(const char* Qualifiers, const char* Device, const char* Interface)
257: {
258:     xTEDSQualifierList QualList;
259:     if(Device==NULL || strcmp(Device,m_strDeviceName)==0)
260:     {
261:         QualList.Parse(Qualifiers);
262:         return m_TreeRoot.AllRegInfo(QualList, Interface);
263:     }
264:     else
265:         return NULL;
266: }
267:
268: MessageDef* xTEDS::RegexRegInfo(const char* Pattern, const char* Qualifiers, const char*
Device, const char* Interface)
269: {
270:     xTEDSQualifierList QualList;
271:     if(Device==NULL || strcmp(Device,m_strDeviceName)==0)
272:     {
273:         QualList.Parse(Qualifiers);
274:         return m_TreeRoot.RegexRegInfo(Pattern, QualList, Interface);
275:     }
276:     else
277:         return NULL;

```

```

278: }
279:
280: SDMMMessage_ID xTEDS::getServiceDataMsgID(const SDMMMessage_ID& CommandID) const
281: {
282:     return m_TreeRoot.GetServiceDataID(CommandID);
283: }
284:
285: SDMMMessage_ID xTEDS::getServiceFaultMsgID(const SDMMMessage_ID& CommandID) const
286: {
287:     return m_TreeRoot.GetServiceFaultID(CommandID);
288: }
289:
290: SDMMMessage_ID xTEDS::getCommandFaultMsgID(const SDMMMessage_ID& CommandID) const
291: {
292:     return m_TreeRoot.GetCommandFaultID(CommandID);
293: }
294:
295: SDMMMessage_ID xTEDS::getNotificationFaultMsgID(const SDMMMessage_ID& DataID) const
296: {
297:     return m_TreeRoot.GetNotificationFaultID(DataID);
298: }
299:
300: bool xTEDS::isCommandIdValid(const SDMMMessage_ID& RequestedId) const
301: {
302:     return m_TreeRoot.IsCommandIdValid(RequestedId);
303: }
304:
305: bool xTEDS::isServiceIdValid(const SDMMMessage_ID& RequestedId) const
306: {
307:     return m_TreeRoot.IsServiceIdValid(RequestedId);
308: }
309:
310: VariableDef* xTEDS::getVarInfo(const char* strVariableName, const SDMMMessage_ID&
idInterface) const
311: {
312:     return m_TreeRoot.VarInfoRequest(strVariableName, idInterface);
313: }
314:

```

File: sdm/common/xTEDS/xTEDSWrapperList.cpp

```
1: #include "xTEDSWrapperList.h"
2: #include "MessageDef.h"
3: #include "xTEDSRequest.h"
4:
5: xTEDSWrapperList::xTEDSWrapperList():Head(NULL)
6: {
7: }
8:
9: xTEDSWrapperList::~~xTEDSWrapperList()
10: {
11: DeleteList();
12: }
13:
14: void xTEDSWrapperList::DeleteList()
15: {
16: for (xTEDSWrapperListNode* Cur = Head; Cur!= NULL; )
17: {
18:     xTEDSWrapperListNode* Temp = Cur->Next;
19:     delete Cur->Data;
20:     delete Cur;
21:     Cur = Temp;
22: }
23: Head = NULL;
24: }
25:
26: void xTEDSWrapperList::AddItem(xTEDSWrapper* Item)
27: {
28: xTEDSWrapperListNode* NewNode = new xTEDSWrapperListNode();
29: NewNode->Data = Item;
30:
31: if (Head == NULL)
32:     Head = NewNode;
33: else
34: {
35:     xTEDSWrapperListNode* Cur;
36:     for (Cur = Head; Cur->Next != NULL; Cur = Cur->Next)
37:         ;
38:     Cur->Next = NewNode;
39: }
```

```

40: }
41:
42: MessageDef* xTEDSWrapperList::RegInfo(const char* Name, const xTEDSQualifierList&
Qualifiers, const char* Interface) const
43: {
44: MessageDef* Def = NULL;
45: for (xTEDSWrapperListNode* Cur = Head; Cur != NULL; Cur = Cur->Next)
46: {
47:     if (Cur->Data != NULL && Cur->Data->RegInfoMatch(Name, Qualifiers, Interface))
48:     {
49:         if (Def == NULL)
50:             Def = Cur->Data->RegInfo(Name);
51:         else
52:             Def->Join(Cur->Data->RegInfo(Name));
53:     }
54: }
55: return Def;
56: }
57:
58: MessageDef* xTEDSWrapperList::RegexRegInfo(const char* Pattern, const xTEDSQualifierList&
Qualifiers, const char* Interface) const
59: {
60: MessageDef* Def = NULL;
61: for (xTEDSWrapperListNode* Cur = Head; Cur != NULL; Cur = Cur->Next)
62: {
63:     if (Cur->Data != NULL && Cur->Data->RegexMatch(Pattern, Qualifiers, Interface))
64:     {
65:         if (Def == NULL)
66:             Def = Cur->Data->RegInfo(NULL);
67:         else
68:             Def->Join(Cur->Data->RegInfo(NULL));
69:     }
70: }
71: return Def;
72: }
73:
74: SDMMMessage_ID xTEDSWrapperList::GetFaultID(const SDMMMessage_ID& MessageID) const
75: {
76: for (xTEDSWrapperListNode* Cur = Head; Cur != NULL; Cur = Cur->Next)
77: {
78:     if (Cur->Data != NULL && Cur->Data->ContainsMessage(MessageID))

```



```

79:     {
80:         // There should only be one instance of this message id, so stop searching
81:         return Cur->Data->GetFaultMessageID();
82:     }
83: }
84: return SDMMMessage_ID(' \0', ' \0');
85: }
86:
87: SDMMMessage_ID xTEDSWrapperList::GetDataID(const SDMMMessage_ID& MessageID) const
88: {
89: for (xTEDSWrapperListNode* Cur = Head; Cur != NULL; Cur = Cur->Next)
90: {
91:     if      (Cur->Data      !=      NULL      &&      Cur->Data->GetType()      ==
xTEDSWrapper::WRAPPER_REQUEST)
92:     {
93:         xTEDSRequest* RequestData = static_cast<xTEDSRequest*>(Cur->Data);
94:
95:         if (RequestData->ContainsMessage(MessageID))
96:             return RequestData->GetDataMessageID();
97:     }
98: }
99: return SDMMMessage_ID(' \0', ' \0');
100: }
101:
102: bool xTEDSWrapperList::ContainsMessage(const SDMMMessage_ID& RequestedId) const
103: {
104:     for (xTEDSWrapperListNode* Cur = Head; Cur != NULL; Cur = Cur->Next)
105:     {
106:         if (Cur->Data != NULL && Cur->Data->ContainsMessage(RequestedId))
107:             return true;
108:     }
109:     return false;
110: }
111:
112: #ifndef REMOVE_DEBUG_OUTPUT
113: void xTEDSWrapperList::PrintDebug() const
114: {
115:     for (xTEDSWrapperListNode* Cur = Head; Cur != NULL; Cur = Cur->Next)
116:     {
117:         if (Cur->Data != NULL)
118:             Cur->Data->PrintDebug();

```

```
119:  }  
120: }  
121: #endif
```

File: sdm/common/xTEDS/xTEDSOrientationList.h

```
1: #ifndef __SDM_XTEDS_ORIENTATION_H_
2: #define __SDM_XTEDS_ORIENTATION_H_
3:
4: #include <stdlib.h>
5: #include "xTEDSOrientationItem.h"
6: extern "C"
7: {
8: #include "xTEDSParser.h"
9: }
10:
11: struct xTEDSOrientationItemNode
12: {
13: xTEDSOrientationItem OrientationData;
14: xTEDSOrientationItemNode *Next;
15: // The below fixes compile warnings
16: xTEDSOrientationItemNode():OrientationData(),Next(NULL) {}
17: xTEDSOrientationItemNode(const xTEDSOrientationItemNode&);
18: xTEDSOrientationItemNode& operator=(const xTEDSOrientationItemNode&);
19: };
20:
21: class xTEDSOrientationList
22: {
23: public:
24: xTEDSOrientationList();
25: ~xTEDSOrientationList();
26: xTEDSOrientationList(const xTEDSOrientationList&);
27: xTEDSOrientationList& operator=(const xTEDSOrientationList&);
28:
29: void AddOrientation(const orientation*);
30: void SetOrientation(const orientation*);
31:
32: void VarInfoRequest(char* InfoBufferOut, size_t BufferLength) const;
33: private:
34: xTEDSOrientationItemNode *Head;
35: void DeleteList();
36: };
37:
38: #endif
39:
```

File: sdm/common/xTEDS/xTEDSOption.cpp

```
1: #include "xTEDSOption.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10:
xTEDSOption::xTEDSOption():m_strName(NULL),m_strDescription(NULL),m_strValue(NULL),m_str
Alarm(NULL)
11: {
12: }
13:
14:          /*xTEDSOption::xTEDSOption(const          xTEDSOption&
b):m_strName(NULL),m_strDescription(NULL),m_strValue(NULL),m_strAlarm(NULL)
15: {
16: if(m_strName!=NULL) free(m_strName);
17: m_strName = strdup(b.m_strName);
18: if(m_strDescription!=NULL) free(m_strDescription);
19: m_strDescription = strdup(b.m_strDescription);
20: if(m_strValue!=NULL) free(m_strValue);
21: m_strValue = strdup(b.m_strValue);
22: if(m_strAlarm!=NULL) free(m_strAlarm);
23: m_strAlarm = strdup(b.m_strAlarm);
24: }
25:
26: xTEDSOption& xTEDSOption::operator=(const xTEDSOption& b)
27: {
28: if(m_strName!=NULL) free(m_strName);
29: m_strName = strdup(b.m_strName);
30: if(m_strDescription!=NULL) free(m_strDescription);
31: m_strDescription = strdup(b.m_strDescription);
32: if(m_strValue!=NULL) free(m_strValue);
33: m_strValue = strdup(b.m_strValue);
34: if(m_strAlarm!=NULL) free(m_strAlarm);
35: m_strAlarm = strdup(b.m_strAlarm);
36: return *this;
37: }
```

```

38: */
39:
40: xTEDSOOption::~xTEDSOOption()
41: {
42: if(m_strName!=NULL) free(m_strName);
43: if(m_strDescription!=NULL) free(m_strDescription);
44: if(m_strValue!=NULL) free(m_strValue);
45: if(m_strAlarm!=NULL) free(m_strAlarm);
46: }
47:
48: void xTEDSOOption::setName(const char* new_name)
49: {
50: if (new_name == NULL) return;
51: if(m_strName!=NULL) free(m_strName);
52: m_strName = strdup(new_name);
53: }
54:
55: void xTEDSOOption::setDescription(const char* new_description)
56: {
57: if (new_description == NULL) return;
58: if(m_strDescription!=NULL) free(m_strDescription);
59: m_strDescription = strdup(new_description);
60: }
61:
62: void xTEDSOOption::setValue(const char* new_value)
63: {
64: if (new_value == NULL) return;
65: if(m_strValue!=NULL) free(m_strValue);
66: m_strValue = strdup(new_value);
67: }
68:
69: void xTEDSOOption::setAlarm(const char* new_alarm)
70: {
71: if (new_alarm == NULL) return;
72: if(m_strAlarm!=NULL) free(m_strAlarm);
73: m_strAlarm = strdup(new_alarm);
74: }
75:
76: void xTEDSOOption::setOption(const curveoption* Option)
77: {
78: if (Option == NULL) return;

```

```

79: setName(Option->name);
80: setValue(Option->value);
81: setDescription(Option->description);
82: setAlarm(Option->alarm);
83: }
84:
85: void xTEDSOption::VarInfoRequest( char* InfoBufferOut, size_t BufferSize ) const
86: {
87:     const unsigned int MAX_BUF_SIZE = 512;
88:     char Buf[MAX_BUF_SIZE], TempBuf[MAX_BUF_SIZE];
89:
90:     if (m_strName == NULL || m_strValue == NULL) return ;
91:
92:     snprintf(Buf, sizeof(Buf), "<Option name= \"%s \" value= \"%s \"",m_strName,m_strValue);
93:
94:     if (m_strDescription != NULL)
95:     {
96:         snprintf(TempBuf, sizeof(TempBuf), " description= \"%s \"", m_strDescription);
97:         strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
98:     }
99:     if (m_strAlarm != NULL)
100:     {
101:         snprintf(TempBuf, sizeof(TempBuf), " alarm= \"%s \"", m_strAlarm);
102:         strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
103:     }
104:     strncat(Buf, ">", sizeof(Buf) - strlen(Buf));
105:
106:     strncat(InfoBufferOut, Buf, BufferSize - 1);
107: }
108:

```

File: sdm/common/xTEDS/xTEDSOrientationItem.cpp

```
1: #include <string.h>
2: #include <stdlib.h>
3: #include <stdio.h>
4: #include "xTEDSOrientationItem.h"
5: #ifdef WIN32
6: # include "unistd.h"
7: #endif
8:
9:
xTEDSOrientationItem::xTEDSOrientationItem():m_strAxis(NULL),m_strAngle(NULL),m_strUnits(N
ULL)
10: {
11: }
12:
13: xTEDSOrientationItem::~~xTEDSOrientationItem()
14: {
15: if (m_strAngle != NULL) free(m_strAngle);
16: if (m_strAxis != NULL) free(m_strAxis);
17: if (m_strUnits != NULL) free(m_strUnits);
18: }
19:
20: void xTEDSOrientationItem::SetAxis(const char* NewAxis)
21: {
22: if (NewAxis == NULL) return;
23: if (m_strAxis != NULL) free(m_strAxis);
24: m_strAxis = strdup(NewAxis);
25: }
26:
27: void xTEDSOrientationItem::SetAngle(const char* NewAngle)
28: {
29: if (NewAngle == NULL) return;
30: if (m_strAngle != NULL) free(m_strAngle);
31: m_strAngle = strdup(NewAngle);
32: }
33:
34: void xTEDSOrientationItem::SetUnits(const char* NewUnits)
35: {
36: if (NewUnits == NULL) return;
37: if (m_strUnits != NULL) free(m_strUnits);
```

```

38: m_strUnits = strdup(NewUnits);
39: }
40:
41: void xTEDSOrientationItem::SetOrientation(const orientation* NewOrientation)
42: {
43: if (NewOrientation == NULL) return;
44: SetAxis(NewOrientation->axis_value);
45: SetAngle(NewOrientation->angle_value);
46: SetUnits(NewOrientation->units_value);
47: }
48:
49: void xTEDSOrientationItem::VarInfoRequest(char* InfoBufferOut, size_t BufferLength) const
50: {
51: const int MAX_BUF_SIZE = 512;
52: char Buf[MAX_BUF_SIZE];
53:
54: if (m_strAxis == NULL || m_strAngle == NULL || m_strUnits == NULL)
55:     return ;
56:
57: snprintf(Buf, sizeof(Buf), "<Orientation axis= \"%s \" angle= \"%s \" units= \"%s \" />", m_strAxis,
m_strAngle, m_strUnits);
58:
59: strncat(InfoBufferOut, Buf, BufferLength - 1);
60: }

```


File: sdm/common/xTEDS/xTEDSCommandMsg.h

```
1: #ifndef __SDM_XTEDS_COMMAND_MSG_H_
2: #define __SDM_XTEDS_COMMAND_MSG_H_
3:
4: #include "xTEDSMessage.h"
5: #include "xTEDSVariable.h"
6: #include "MessageDef.h"
7: #include "xTEDSVariableList.h"
8:
9:
10: class xTEDSCommandMsg:public xTEDSMessage
11: {
12: public:
13: xTEDSCommandMsg();
14: virtual ~xTEDSCommandMsg();
15:
16: MessageDef* RegInfo(void) const;
17: bool RegInfoMatch(const char* name, const xTEDSQualifierList& , const char* interface) const;
18: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
Interface) const;
19: bool SetCommandMsg(const command_message* Command, const xTEDSVariableList&
VariableList);
20:
21: virtual void PrintDebug() const;
22: private:
23: bool MatchesQualifier(const xTEDSQualifierList&) const;
24: void xTEDSInfo(char* InfoBufferOut, int BufferSize) const;
25: };
26:
27: #endif
```

File: sdm/common/xTEDS/xTEDSVerification.cpp

```
1: #include "xTEDSVerification.h"
2:
3: bool xTEDSVerification::VerifyQualifiers(const char* ItemName, const char* Interface,
qualifier_type* QualifierList)
4: {
5:     if (QualifierList == NULL)
6:         return false;
7:
8:     for (qualifier_type* CurQual = QualifierList; CurQual != NULL; CurQual = CurQual->next)
9:     {
10:         if (CurQual->name == NULL)
11:         {
12:             printf("Error: The Qualifier name must be set for %s in Interface %s\n",ItemName,Interface);
13:             return false;
14:         }
15:         if (CurQual->value == NULL)
16:         {
17:             printf("Error: The Qualifier %s must have the value field set in %s in Interface %s\n",CurQual->name,ItemName,Interface);
18:             return false;
19:         }
20:     }
21:     return true;
22: }
23:
24: bool xTEDSVerification::VerifyOrientation(const variable* var)
25: {
26:     if (var == NULL || var->orientation_data == NULL)
27:         return false;
28:
29:     if (var->orientation_data->axis_value == NULL || var->orientation_data->angle_value == NULL ||
var->orientation_data->units_value == NULL)
30:     {
31:         printf("Error: Variable %s is missing a required orientation attribute. \n",var->name);
32:         return false;
33:     }
34:     return true;
35: }
36:
```

```

37: bool xTEDSVerification::VerifyLocation(const variable* var, const location* loc)
38: {
39: if (var == NULL || loc == NULL)
40:     return false;
41:
42: if (loc->x_value == NULL || loc->y_value == NULL || loc->z_value == NULL)
43: {
44:     printf("Error: Variable %s is missing a location component. \n",var->name);
45:     return false;
46: }
47: if (loc->units == NULL)
48: {
49:     printf("Error: Variable %s is missing the units field in its location element. \n",var->name);
50:     return false;
51: }
52: return true;
53: }
54:
55: bool xTEDSVerification::VerifyCurve (const variable* var)
56: {
57: if (var == NULL || var->curves == NULL || var->interface_name == NULL || var->name == NULL)
58:     return false;
59:
60: if (var->curves->name == NULL)
61: {
62:     printf("Error: The Curve name must be set for Variable %s in Interface %s! \n",var->name, var-
>interface_name);
63:     return false;
64: }
65: if (var->curves->coefs == NULL)
66: {
67:     printf("Error: The Curve %s must have at least 1 Coef in Interface %s! \n",var->curves-
>name,var->interface_name);
68:     return false;
69: }
70: for (coef* CurCoef = var->curves->coefs; CurCoef != NULL; CurCoef = CurCoef->next)
71: {
72:     if (CurCoef->exponent == NULL)
73:     {
74:         printf("Error: The Coef must have the exponent field set for Curve %s in Interface %s!
\n",var->curves->name,var->interface_name);
75:         return false;

```

```

76:     }
77:     if (CurCoef->value == NULL)
78:     {
79:         printf("Error: The Coef %s must have the value field set for Curve %s in Interface %s.
\n",CurCoef->exponent,var->curves->name,var->interface_name);
80:         return false;
81:     }
82: }
83: return true;
84: }
85:
86: bool xTEDSVerification::VerifyDrange (const variable* var)
87: {
88: if (var == NULL || var->dranges == NULL || var->interface_name == NULL || var->name == NULL)
89:     return false;
90:
91: // Do some verification checking on the drange and option values
92: if (var->dranges->name == NULL)
93: {
94:     printf("Error: The Drange name must be set for Variable %s in Interface %s. \n",var->name,var-
>interface_name);
95:     return false;
96: }
97: // Be sure there is at least one option
98: if (var->dranges->options == NULL)
99: {
100:     printf("Error: The Drange %s must have at least 1 Option in Interface %s. \n",var->dranges-
>name,var->interface_name);
101:     return false;
102: }
103: // Check to be sure all options have at least a name and value
104: for (curveoption* CurOption = var->dranges->options; CurOption != NULL; CurOption =
CurOption->next)
105: {
106:     if (CurOption->name == NULL)
107:     {
108:         printf("Error: The option name must be set for Drange %s in Interface %s. \n",var-
>dranges->name,var->interface_name);
109:         return false;
110:     }
111:     else if (CurOption->value == NULL)
112:     {

```

```

113:         printf("Error: The option %s must have the value field set for Drange %s in Interface %s.
\n",CurOption->name,var->dranges->name,var->interface_name);
114:         return false;
115:     }
116: }
117: return true;
118: }
119:
120: bool xTEDSVerification::VerifyVarRefs(const char* MsgName, var_ref* VarRefs)
121: {
122:     if (MsgName == NULL || VarRefs == NULL)
123:         return false;
124:
125:     for (var_ref* Cur = VarRefs; Cur != NULL; Cur = Cur->next)
126:     {
127:         if (Cur->name == NULL)
128:         {
129:             printf("Variable reference name not specified for message %s. \n",MsgName);
130:             return false;
131:         }
132:     }
133:     return true;
134: }
135:
136: bool xTEDSVerification::VerifyVariable(variable* var)
137: {
138:     if (var == NULL)
139:         return false;
140:
141:     if(var->name == NULL)
142:     {
143:         printf("Error: Variable is missing the name field! \n");
144:         return false;
145:     }
146:     if(var->kind == NULL)
147:     {
148:         printf("Error: Variable %s is missing the kind field! \n",var->name);
149:         return false;
150:     }
151:     if(var->interface_name == NULL)
152:     {

```

```

153:     printf("Error: The Variable %s is not within an interface \n",var->name);
154:     return false;
155: }
156: if(var->interface_id == NULL)
157: {
158:     printf("Error: The Variable %s is not within an interface \n",var->name);
159:     return false;
160: }
161: if(var->format != NULL)
162: {
163:     if( strcmp(var->format,"INT08") !=0 &&
164:        strcmp(var->format,"UINT08") !=0 &&
165:        strcmp(var->format,"INT16") !=0 &&
166:        strcmp(var->format,"UINT16") !=0 &&
167:        strcmp(var->format,"INT32") !=0 &&
168:        strcmp(var->format,"UINT32") !=0 &&
169:        strcmp(var->format,"FLOAT32") !=0 &&
170:        strcmp(var->format,"FLOAT64") !=0 )
171:     {
172:         printf("Error: The format field value %s for Variable %s does not match any defined type
in Interface %s! \n",var->format,var->name,var->interface_name);
173:         return false;
174:     }
175: }
176: else
177: {
178:     printf("Error: Variable %s is missing the format field in Interface %s! \n",var->name,var-
>interface_name);
179:     return false;
180: }
181: if (var->qualifiers != NULL)
182:     if (!VerifyQualifiers(var->name, var->interface_name, var->qualifiers))
183:         return false;
184:
185: if(var->dranges!=NULL)
186:     if (!VerifyDrange (var))
187:         return false;
188:
189: if(var->curves!=NULL)
190:     if (!VerifyCurve (var))
191:         return false;

```

```

192:
193:  if (var->location_data != NULL)
194:      if (!VerifyLocation (var, var->location_data))
195:          return false;
196:
197:  if (var->orientation_data != NULL)
198:      if (!VerifyOrientation (var))
199:          return false;
200:  return true;
201: }
202:
203: bool xTEDSVerification::VerifyDataMsg(const data_msg* dat)
204: {
205:     if (dat == NULL)
206:         return false;
207:
208:     if(dat->name == NULL)
209:     {
210:         printf("Error: The DataMsg or DataReplyMsg is missing the name field! \n");
211:         return false;
212:     }
213:     if(dat->id == NULL)
214:     {
215:         printf("Error: The DataMsg or DataReplyMsg %s is missing the id field! \n",dat->name);
216:         return false;
217:     }
218:     if(dat->interface_name == NULL)
219:     {
220:         printf("Error: The DataMsg %s is not within an interface \n",dat->name);
221:         return false;
222:     }
223:     if(dat->interface_id == NULL)
224:     {
225:         printf("Error: The DataMsg %s is not within an interface \n",dat->name);
226:         return false;
227:     }
228:     if(dat->msg_arrival != NULL)
229:     {
230:         if( strcmp(dat->msg_arrival,"EVENT") != 0 &&
231:            strcmp(dat->msg_arrival,"PERIODIC") != 0 )
232:         {

```

```

223:         printf("Error: The msgArrival type %s for DataMsg %s does not match any defined
value in Interface %s \n",dat->msg_arrival,dat->name,dat->interface_name);
224:         return false;
225:     }
226: }
227:
228: // Add any qualifiers
229: if (dat->qualifiers != NULL)
230:     if (!VerifyQualifiers(dat->name, dat->interface_name, dat->qualifiers))
231:         return false;
232:
233: if (dat->variables != NULL)
234: {
235:     if (!VerifyVarRefs(dat->name, dat->variables))
236:         return false;
237: }
238:
239: return true;
240: }
241:
242: bool xTEDSVerification::VerifyCommandMsg(const cmd_msg* cmd)
243: {
244:     if (cmd == NULL)
245:         return false;
246:
247:     if(cmd->name == NULL)
248:     {
249:         printf("Error: The CommandMsg is missing the name field! \n");
250:         return false;
251:     }
252:     if(cmd->id == NULL)
253:     {
254:         printf("Error: The CommandMsg %s is missing the id field! \n",cmd->name);
255:         return false;
256:     }
257:
258:     if(cmd->interface_name == NULL)
259:     {
260:         printf("Error: The CommandMsg %s is not within an interface \n",cmd->name);
261:         return false;
262:     }
263: }

```



```

273:  if(cmd->interface_id == NULL)
274:  {
275:      printf("Error: The CommandMsg %s is not within an interface \n",cmd->name);
276:      return false;
277:  }
278:
279:  // Add any qualifiers
280:  if (cmd->qualifiers != NULL)
281:  {
282:      if (VerifyQualifiers(cmd->name, cmd->interface_name, cmd->qualifiers) == false)
283:          return false;
284:  }
285:
286:  if (cmd->variables != NULL)
287:  {
288:      if (!VerifyVarRefs(cmd->name, cmd->variables))
289:          return false;
290:  }
291:  return true;
292: }
293:
294: bool xTEDSVerification::VerifyFaultMsg(const fault_msg* fault)
295: {
296:     if (fault == NULL)
297:         return false;
298:
299:     if(fault->id == NULL)
300:     {
301:         printf("Fault message is missing its id field. \n");
302:         return false;
303:     }
304:     if(fault->name == NULL)
305:     {
306:         printf("Fault message is missing the name field. \n");
307:         return false;
308:     }
309:     if(fault->interface_name == NULL)
310:     {
311:         printf("Fault message %s is not within an interface. \n",fault->name);
312:         return false;
313:     }

```

```

314:   if(fault->interface_id == NULL)
315:   {
316:       printf("Could not find the interface id of fault message %s. \n",fault->name);
317:       return false;
318:   }
319:   if(fault->qualifiers != NULL)
320:   {
321:       if (!VerifyQualifiers (fault->name, fault->interface_name, fault->qualifiers))
322:           return false;
323:   }
324:   if (fault->variables != NULL)
325:   {
326:       if (!VerifyVarRefs(fault->name, fault->variables))
327:           return false;
328:   }
329:
330:   return true;
331: }
332:
333: bool xTEDSVerification::VerifyInterface(const interface* Interface)
334: {
335:     if (Interface == NULL)
336:         return false;
337:
338:     if(Interface->name == NULL)
339:     {
340:         printf("Error: The Interface has a missing name field that is required! \n");
341:         return false;
342:     }
343:     if(Interface->id == NULL || strcmp(Interface->id,"0") == 0)
344:     {
345:         printf("Error: Interface %s has either a interface id of 0 or does not have an interface id!
\n",Interface->name);
346:         return false;
347:     }
348:     return true;
349: }
350:

```

File: sdm/common/xTEDS/xTEDSQualifierList.cpp

```
1: #include "xTEDSQualifierList.h"
2: #include "xTEDSQualifier.h"
3: #include "../Exception/SDMBadIndexException.h"
4:
5: #include <stdlib.h>
6: #include <string.h>
7:
8: struct xTEDSQualifierListNode* copyList(struct xTEDSQualifierListNode* list, struct
xTEDSQualifierListNode** tail)
9: {
10: struct xTEDSQualifierListNode* p;
11: struct xTEDSQualifierListNode* head;
12: head = (struct xTEDSQualifierListNode*)malloc(sizeof(struct xTEDSQualifierListNode));
13: p = head;
14: for(struct xTEDSQualifierListNode* cur=list; cur!=NULL; cur=cur->next)
15: {
16:     p->data = cur->data;
17:     if(cur->next!=NULL)
18:     {
19:         p->next = (struct xTEDSQualifierListNode*)malloc(sizeof(struct
xTEDSQualifierListNode));
20:     }
21:     else
22:     {
23:         p->next = NULL;
24:         *tail = p;
25:     }
26:     p = p->next;
27: }
28: return head;
29: }
30:
31: void deleteList(struct xTEDSQualifierListNode* p)
32: {
33: if(p==NULL) return;
34: deleteList(p->next);
35: if(p->data!=NULL)
36:     delete(p->data);
37: free(p);
```

```

38: }
39:
40: xTEDSQualifierList::xTEDSQualifierList():itemCount(0),head(NULL),tail(NULL)
41: {}
42:
43: xTEDSQualifierList::xTEDSQualifierList(const xTEDSQualifierList&
b):itemCount(0),head(NULL),tail(NULL)
44: {
45: head = copyList(b.head,&tail);
46: }
47:
48: xTEDSQualifierList::~xTEDSQualifierList()
49: {
50: deleteList(head);
51: }
52:
53: void xTEDSQualifierList::addQualifier(const xTEDSQualifier &NewQualifier)
54: {
55: struct xTEDSQualifierListNode* p = (struct xTEDSQualifierListNode*)malloc(sizeof(struct
xTEDSQualifierListNode));
56: p->data = new xTEDSQualifier(NewQualifier);
57: p->next = NULL;
58: if(tail == NULL)
59: {
60: head = p;
61: tail = p;
62: }
63: else
64: {
65: tail->next = p;
66: tail = p;
67: }
68: itemCount++;
69: }
70:
71: xTEDSQualifierList& xTEDSQualifierList::operator=(const xTEDSQualifierList& b)
72: {
73: deleteList(head);
74: head = copyList(b.head,&tail);
75: return *this;
76: }

```

```

77:
78: void xTEDSQualifierList::Parse(const char* quallist)
79: {
80: char quals[256];
81: int start = 0,end = 0;
82: size_t count = 0;
83: bool equalfound = false;
84: bool parseValid = false;
85:
86: quals[0] = 0;
87: count = strlen(quallist);
88: for(unsigned int i = 0; i < count; i++)
89: {
90:     if((quallist[i] != '<' && quallist[i] != '>') && end >= start)
91:     {
92:         start = i;
93:     }
94:     if(start > end && equalfound == false && quallist[i] == '=')
95:     {
96:         equalfound = true;
97:         i++;
98:         while(quallist[i] == ' ' && i < count)
99:         {
100:             i++;
101:         }
102:     }
103:     if(equalfound == true && (quallist[i] == ' ' || quallist[i] == '/'))
104:     {
105:         end = i;
106:     }
107:     if(end > start)
108:     {
109:         xTEDSQualifier m_qual;
110:         strncpy(quals,&quallist[start],end-start);
111:         quals[end-start] = 0;
112:         parseValid = m_qual.Parse(quals);
113:         if(parseValid == true)
114:         {
115:             addQualifier(m_qual);
116:         }
117:     }
    else

```

```

118:         {
119:             printf("!!! Improperly formatted Qualifier List !!! \n");
120:         }
121:         equalfound = false;
122:         parseValid = false;
123:         start = end;
124:     }
125: }
126: }
127:
128: void xTEDSQualifierList::QualifierInfoRequest( char* InfoBufferOut, size_t BufferSize )
129: {
130:     char Buf[MSG_DEF_SIZE];
131:     size_t BufLength = 0;
132:
133:     Buf[0] = '\0';
134:     if(head != NULL)
135:     {
136:         // Add the first qualifier
137:         head->data->QualifierInfoRequest(Buf, sizeof(Buf));
138:         BufLength = strlen(Buf);
139:
140:         for(struct xTEDSQualifierListNode* cur=head->next; cur != NULL; cur=cur->next)
141:         {
142:             // Add the formatting whitespace
143:             strncat (Buf, " \n \t \t", sizeof(Buf) - BufLength);
144:             BufLength += 3;
145:
146:             // Add the current qualifier text
147:             cur->data->QualifierInfoRequest(Buf + BufLength, sizeof(Buf) - BufLength);
148:             BufLength = strlen(Buf);
149:         }
150:     }
151:     strncat(InfoBufferOut, Buf, BufferSize);
152: }
153:
154: int xTEDSQualifierList::Size() const
155: {
156:     return itemCount;
157: }
158:

```

```

159: bool xTEDSQualifierList::isEmpty() const
160: {
161:     return (itemCount == 0);
162: }
163:
164: // Zero-based index
165: const xTEDSQualifier& xTEDSQualifierList::operator[](int index) const
166: {
167:     if (index < 0 || index >= itemCount)
168:         throw SDMBadIndexException("xTEDSQualifierList:: operator[] index out of bounds.");
169:
170:     int Count = 0;
171:     xTEDSQualifierListNode* Cur;
172:     for (Cur = head; Cur != NULL && Count < index; Cur = Cur->next)
173:         ;
174:
175:     if (Cur == NULL && Cur->data == NULL)
176:         throw SDMBadIndexException("xTEDSQualifierList:: no data to return.");
177:
178:     return *(Cur->data);
179: }

```

File: sdm/common/xTEDS/xTEDSVariableList.cpp

```
1: #include "xTEDSVariableList.h"
2: #include "xTEDSVariable.h"
3: #include "MessageDef.h"
4:
5: #include <stdlib.h>
6: #include <string.h>
7: #include <stdio.h>
8:
9: /*
10: struct xTEDSItemListNode* copyList(struct xTEDSItemListNode* list,struct xTEDSItemListNode**
tail)
11: {
12: struct xTEDSItemListNode* p;
13: struct xTEDSItemListNode* head;
14: if(list == NULL)
15:     return NULL;
16: head = (struct xTEDSItemListNode*)malloc(sizeof(struct xTEDSItemListNode));
17: p = head;
18: for(struct xTEDSItemListNode* cur=list;cur!=NULL;cur=cur->next)
19: {
20:     p->data = cur->data;
21:     if(cur->next!=NULL)
22:     {
23:         p->next = (struct xTEDSItemListNode*)malloc(sizeof(struct xTEDSItemListNode));
24:     }
25:     else
26:     {
27:         p->next = NULL;
28:         *tail = p;
29:     }
30:     p = p->next;
31: }
32: return head;
33: }*/
34:
35: xTEDSVariableList::xTEDSVariableList():head(NULL),tail(NULL)
36: {}
37: /*
38: xTEDSVariableList::xTEDSVariableList(const xTEDSVariableList& b):head(NULL),tail(NULL)
```



```

39: {
40: head = copyList(b.head,&tail);
41: }
42: */
43:
44: /*
45: * Delete only the xTEDSVariableListNode objects, the xTEDSVariables are handled outside
46: * this class.
47: */
48: xTEDSVariableList::~~xTEDSVariableList()
49: {
50: if (!IsEmpty())
51: {
52:     for (xTEDSVariableListNode* Cur = head; Cur != NULL; )
53:     {
54:         xTEDSVariableListNode* Temp = Cur->next;
55:         delete Cur;
56:         Cur = Temp;
57:     }
58:     head = tail = NULL;
59: }
60: }
61:
62: /*
63: * Delete everything in the list. Both the xTEDSVariableListNode objects, and the xTEDSVariables
64: * they hold a reference to.
65: */
66: void xTEDSVariableList::DeleteItems()
67: {
68: for (xTEDSVariableListNode* Cur = head; Cur != NULL; )
69: {
70:     xTEDSVariableListNode* Temp = Cur->next;
71:     delete Cur->data;
72:     delete Cur;
73:     Cur = Temp;
74: }
75: head = tail = NULL;
76: }
77:
78: void xTEDSVariableList::addItem(const xTEDSVariable* data)
79: {

```

```

80: xTEDSVariableListNode* p = new xTEDSVariableListNode(data);
81:
82: if(tail == NULL)
83: {
84:     head = p;
85:     tail = p;
86: }
87: else
88: {
89:     tail->next = p;
90:     tail = p;
91: }
92: }
93: /*
94: xTEDSVariableList& xTEDSVariableList::operator=(const xTEDSItemList& b)
95: {
96: deleteList(head);
97: head = copyList(b.head,&tail);
98: return *this;
99: }*/
100:
101: void xTEDSVariableList::VarReqReg(char* InfoBufferOut, int BufferSize) const
102: {
103:     char Buf[MSG_DEF_SIZE];
104:     int BufLength = 0;
105:     int StartByte = 0;
106:
107:     Buf[0] = '\0';
108:     for(xTEDSVariableListNode* cur=head;cur!=NULL;cur=cur->next)
109:     {
110:         BufLength = strlen(Buf);
111:         strncat(Buf, " ", sizeof(Buf) - BufLength);
112:
113:         BufLength += 1;
114:         cur->data->VarRegInfo(Buf + BufLength, sizeof(Buf) - BufLength, StartByte);
115:     }
116:     char TempBuf[128];
117:     sprintf(TempBuf," length= \"%d \",StartByte);
118:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
119:
120:     strncat(InfoBufferOut, Buf, BufferSize - 1);

```

```

121: }
122:
123: void xTEDSVariableList::VarRef(char* InfoBufferOut, int BufferSize) const
124: {
125:     char Buf[MSG_DEF_SIZE];
126:     int BufLength = 0;
127:
128:     Buf[0] = '\0';
129:     for(struct xTEDSVariableListNode* cur=head; cur != NULL; cur=cur->next)
130:     {
131:         BufLength = strlen(Buf);
132:         if (cur!=head)
133:         {
134:             strncat(Buf, "\n\t\t", sizeof(Buf) - BufLength);
135:             BufLength += 3;
136:         }
137:         cur->data->VarRef(Buf + BufLength, sizeof(Buf) - BufLength);
138:     }
139:     strncat(InfoBufferOut, Buf, BufferSize - 1);
140: }
141:
142: MessageDef* xTEDSVariableList::RegInfo() const
143: {
144:     /*char buf[MSG_DEF_SIZE];
145:     MessageDef* m_def_a = NULL;
146:     MessageDef* m_def_b;
147:
148:     memset(buf,0,MSG_DEF_SIZE);
149:     if(head == NULL)
150:         return NULL;
151:     for(struct xTEDSItemListNode* cur=head;cur!=NULL;cur=cur->next)
152:     {
153:         if(cur==head)
154:         {
155:             m_def_a = cur->data->RegInfo(NULL,NULL); //owner of list already matches
156:         }
157:         else
158:         {
159:             m_def_b = cur->data->RegInfo(NULL,NULL); //owner of list already matches
160:             if(m_def_a != NULL)
161:             {

```

```

162:         m_def_a->Join(m_def_b);
163:     }
164:     else
165:     {
166:         m_def_a = m_def_b;
167:     }
168:
169:     }
170: }
171: return m_def_a;*/
172: return NULL;
173: }
174:
175: bool xTEDSVariableList::IsEmpty() const
176: {
177:     if(head == NULL)
178:         return true;
179:     return false;
180: }
181:
182: const xTEDSVariable* xTEDSVariableList::Find(const char* name, const char* interface) const
183: {
184:     if (name == NULL) return NULL;
185:
186:     xTEDSVariableListNode* Cur = NULL;
187:     for (Cur = head; Cur != NULL; Cur = Cur->next)
188:     {
189:         if (Cur->data != NULL)
190:         {
191:             if (strcmp(Cur->data->getName(), name)==0)
192:             {
193:                 if (interface == NULL || strcmp(Cur->data->getInterfaceName(), interface)==0)
194:                     return Cur->data;
195:             }
196:         }
197:     }
198:     return NULL;
199: }
200:
201: bool xTEDSVariableList::RegInfoMatch(const char* pname, const xTEDSQualifierList& qualifiers,
const char* interface) const

```

```

202: {
203:     for (xTEDSVariableListNode* Cur = head; Cur != NULL; Cur = Cur->next)
204:     {
205:         if (Cur->data != NULL)
206:         {
207:             if(Cur->data->RegInfoMatch(pname, qualifiers, interface))
208:                 return true;
209:         }
210:     }
211:     return false;
212: }
213:
214: bool xTEDSVariableList::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList,
const char* Interface) const
215: {
216:     for (xTEDSVariableListNode* Cur = head; Cur != NULL; Cur = Cur->next)
217:     {
218:         if (Cur->data != NULL)
219:         {
220:             if(Cur->data->RegexMatch(Pattern, QualList, Interface))
221:                 return true;
222:         }
223:     }
224:     return false;
225: }
226:
227: VariableDef* xTEDSVariableList::VarInfoRequest(const char* VarName, const
SDMMMessage_ID& Interface,
228:          VarInfoMatchType matchType) const
229: {
230:     VariableDef* VarDef = NULL;
231:     for (xTEDSVariableListNode* Cur = head; Cur != NULL; Cur = Cur->next)
232:     {
233:         if (Cur->data != NULL)
234:         {
235:             // If the variable names match
236:             if (((matchType == MATCH_VAR_NAME && strcmp(Cur->data->getName(),
VarName)==0)) ||
237:                 matchType == MATCH_ALL)
238:             {
239:                 // If the interface was not specified(0) or the interface ids match

```

```

240:         if (Interface.getInterface() == 0 || Interface.getInterface() == Cur->data-
>getInterfaceID())
241:         {
242:             if (VarDef == NULL)
243:                 VarDef = Cur->data->VarInfo();
244:             else
245:                 VarDef->Join(Cur->data->VarInfo());
246:         }
247:     }
248: }
249: }
250: return VarDef;
251: }
252:
253: #ifndef REMOVE_DEBUG_OUTPUT
254: void xTEDSVariableList::PrintDebug() const
255: {
256:     for (xTEDSVariableListNode* Cur = head; Cur != NULL; Cur = Cur->next)
257:     {
258:         if (Cur->data != NULL)
259:             Cur->data->PrintDebug();
260:     }
261: }
262: #endif
263:
264:

```

File: sdm/common/xTEDS/xTEDS.l

```
1: %{
2: #define _GNU_SOURCE
3: #include <stdio.h>
4: #include <stdlib.h>
5: #include <string.h>
6:
7: #ifndef WIN32
8: # ifndef strndup
9: #  include "../MemoryUtils.h"
10: # endif
11: #else
12: # include "unistd.h"
13: #endif
14:
15: #include "xTEDS.tab.h"
16: int line_count = 1;
17: %{
18:
19: %option nounput
20:
21: %%
22:
23: "="                {return EQUAL_SY;}
24: ">"                {return CLOSE_SY;}
25: ">"                {return SLASHCLOSE_SY;}
26: "?>"              {return CLOSE_XML_SY;}
27:
28: "<?xml"            {return OPEN_XML_SY;}
29: "</xTEDS"          {return CLOSE_xTEDS_SY;}
30: "<xTEDS"           {return OPEN_xTEDS_SY;}
31: "<Application"     {return OPEN_APP_SY;}
32: "</Application"    {return CLOSE_APP_SY;}
33: "<Variable"        {return OPEN_VAR_SY;}
34: "</Variable"       {return CLOSE_VAR_SY;}
35: "<Drange"          {return OPEN_DRANGE_SY;}
36: "</Drange"         {return CLOSE_DRANGE_SY;}
37: "<Option"          {return OPEN_OPTION_SY;}
38: "</Option"         {return CLOSE_OPTION_SY;}
39: "<Curve"           {return OPEN_CURVE_SY;}
```

```

40: "</Curve"                {return CLOSE_CURVE_SY;}
41: "<Coef"                  {return OPEN_COEFF_SY;}
42: "</Coef"                 {return CLOSE_COEFF_SY;}
43: "<DataMsg"|"<DataReplyMsg"    {return OPEN_DATA_MSG_SY;}
44: "</DataMsg"|"</DataReplyMsg"    {return CLOSE_DATA_MSG_SY;}
45: "<VariableRef"            {return OPEN_VARIABLE_REF_SY;}
46: "</VariableRef"          {return CLOSE_VARIABLE_REF_SY;}
47: "<CommandMsg"            {return OPEN_COMMAND_MSG_SY;}
48: "</CommandMsg"           {return CLOSE_COMMAND_MSG_SY;}
49: "<Device"                 {return OPEN_DEVICE_SY;}
50: "</Device"               {return CLOSE_DEVICE_SY;}
51: "<Interface"             {return OPEN_INTERFACE_SY;}
52: "</Interface"           {return CLOSE_INTERFACE_SY;}
53: "<Command"               {return OPEN_COMMAND_SY;}
54: "</Command"              {return CLOSE_COMMAND_SY;}
55: "<Notification"          {return OPEN_NOTIFICATION_SY;}
56: "</Notification"        {return CLOSE_NOTIFICATION_SY;}
57: "<Request"               {return OPEN_REQUEST_SY;}
58: "</Request"              {return CLOSE_REQUEST_SY;}
59: "<FaultMsg"              {return OPEN_FAULT_MSG_SY;}
60: "</FaultMsg"             {return CLOSE_FAULT_MSG_SY;}
61: "<Qualifier"             {return OPEN_QUALIFIER_SY;}
62: "</Qualifier"           {return CLOSE_QUALIFIER_SY;}
63: "<Location"              {return OPEN_LOCATION_SY;}
64: "</Location"            {return CLOSE_LOCATION_SY;}
65: "<Orientation"           {return OPEN_ORIENTATION_SY;}
66: "</Orientation"         {return CLOSE_ORIENTATION_SY;}
67:
68: "invalidValue"            {return INVALID_VALUE_SY;}
69: /* Because "id" is a substring of "invalidValue", the following pattern will
70: catch a misspelling of "invalidValue" without matching "id". This prevents the
71: parser from incorrectly reducing grammar rules that were intended for "invalidValue"
72: but instead "id" was matched -- specifically with <Variable> attributes.*/
73: [^ \t \n]"id"|"id"[^ \t \n=]    {return BAD_TERMINAL_SY;}
74: "name"                    {return NAME_SY;}
75: "kind"                   {return KIND_SY;}
76: "id"                     {return ID_SY;}
77: "qualifier"              {return QUALIFIER_SY;}
78: "description"            {return DESCRIPTION_SY;}
79: "manufacturerId"         {return MANUFACTURER_ID_SY;}
80: "version"                {return VERSION_SY;}

```


81: "modelId"	{return MODEL_ID_SY;}
82: "versionLetter"	{return VERSION_LETTER_SY;}
83: "serialNumber"	{return SERIAL_NUMBER_SY;}
84: "calibrationDate"	{return CALIBRATION_DATE_SY;}
85: "sensitivityAtReference"	{return SENSITIVITY_AT_REF_SY;}
86: "referenceFrequency"	{return REF_FREQ_SY;}
87: "referenceTemperature"	{return REF_TEMP_SY;}
88: "measurementRange"	{return MEASUREMENT_RANGE_SY;}
89: "electricalOutput"	{return ELECTRICAL_OUTPUT_SY;}
90: "qualityFactor"	{return QUALITY_FACTOR_SY;}
91: "temperatureCoefficient"	{return TEMP_COEFF_SY;}
92: "directionXYZ"	{return DIRECTION_XYZ_SY;}
93: "calDueDate"	{return CAL_DUE_DATE_SY;}
94: "powerRequirements"	{return POWER_REQS_SY;}
95: "value"	{return VALUE_SY;}
96: "alarm"	{return ALARM_SY;}
97: "msgArrival"	{return MSG_ARRIVAL_SY;}
98: "msgRate"	{return MSG_RATE_SY;}
99: "precision"	{return PRECISION_SY;}
100: "rangeMax"	{return RANGE_MAX_SY;}
101: "format"	{return FORMAT_SY;}
102: "accuracy"	{return ACCURACY_SY;}
103: "rangeMin"	{return RANGE_MIN_SY;}
104: "scaleFactor"	{return SCALE_FACTOR_SY;}
105: "units"	{return UNITS_SY;}
106: "defaultValue"	{return DEFAULT_VALUE_SY;}
107: "scaleUnits"	{return SCALE_UNITS_SY;}
108: "length"	{return LENGTH_SY;}
109: "exponent"	{return EXPONENT_SY;}
110: "componentKey"	{return COMPONENT_KEY_SY;}
111: "SPA_U_hub" "spaUHub"	{return SPA_U_HUB_SY;}
112: "SPA_U_port" "spaUPort"	{return SPA_U_PORT_SY;}
113: "extends"	{return EXTENDS_SY;}
114: "memoryMinimum"	{return MEMORY_MINIMUM_SY;}
115: "operatingSystem"	{return OPERATING_SYSTEM_SY;}
116: "pathForAssembly"	{return PATH_FOR_ASSEMBLY_SY;}
117: "pathOnSpacecraft"	{return PATH_ON_SPACECRAFT_SY;}
118: "x"	{return X_SY;}
119: "y"	{return Y_SY;}
120: "z"	{return Z_SY;}
121: "axis"	{return AXIS_SY;}

```

122: "angle"                {return ANGLE_SY;}
123: "encoding"             {return ENCODING_SY;}
124: "standalone"           {return STANDALONE_SY;}
125: "rLow"                 {return R_LOW_SY;}
126: "rHigh"                {return R_HIGH_SY;}
127: "yLow"                 {return Y_LOW_SY;}
128: "yHigh"                {return Y_HIGH_SY;}
129:
130: "xmlns"                 {return XMLNS_SY;}
131: "xmlns:xsi"             {return XMLNS_XSI_SY;}
132: "xsi:schemaLocation"   {return SCHEMA_LOCATION_SY;}
133:
134:
135: \"[^\"]*\"               {xTEDSIval.str = strdup(yytext+1,strlen(yytext)-2);return STRING;}
136: [0-9]*[.][0-9]+          {xTEDSIval.real = atof(yytext); return FLOAT;}
137: [0-9]+                   {xTEDSIval.integer = atoi(yytext); return INT;}
138:
139: "<!--"([^\-]*[^\-])*)*"-->"| "<!-- -->"      {/*ignore xteds comments*/}
140:
141: [ \t]*                   {/*ignore whitespace*/}
142:
143: "\n"                     {line_count++;}
144:
145: .                         {/* catch all, don't print invalid chars*/}
146:
147: %%
148:
149: int yywrap() {return 1;}
150: void xTEDSError(char *s)
151: {
152:     printf("  Syntax error in xTEDS on line %d at token \"%s\".\n",line_count,yytext);
153: }

```

File: sdm/common/xTEDS/xTEDSItemTree.cpp

```
1: #include "xTEDSItemTree.h"
2: #include "xTEDSItem.h"
3: #include "MessageDef.h"
4: #include "xTEDSCommandMsg.h"
5:
6: #include <stdlib.h>
7: #include <string.h>
8: #include <stdio.h>
9:
10:
11: //Note an xTEDSItemTree owns its data items and so should delete them and make deep copies as
needed.
12:
13: xTEDSItemTree::xTEDSItemTree():m_Variables(),m_Commands(),m_Notifications(),m_Requests()
14: {
15: }
16:
17: xTEDSItemTree::~~xTEDSItemTree()
18: {
19: // Variables is the only list the needs to be explicitly freed
20: m_Variables.DeleteItems();
21: }
22:
23: xTEDSItemTree& xTEDSItemTree::operator=(const xTEDSItemTree& Right)
24: {
25: printf("xTEDSItemTree::operator= \n");
26: return *this;
27: }
28:
29: xTEDSItemTree::xTEDSItemTree(const xTEDSItemTree&
Right):m_Variables(),m_Commands(),m_Notifications(),m_Requests()
30: {
31: printf("xTEDSItemTree::Copy constructor \n");
32: }
33:
34: MessageDef* xTEDSItemTree::ExactRegInfo(const char* name, const xTEDSQualifierList&
qualifiers, const char* interface) const
35: {
36: //TODO: Remove the dynamic allocation and return by using copy constructor
37: MessageDef* Result = NULL;
```

```

38:
39: Result = m_Commands.RegInfo(name, qualifiers, interface);
40:
41: if (Result != NULL)
42:     Result->Join(m_Notifications.RegInfo(name, qualifiers, interface));
43: else
44:     Result = m_Notifications.RegInfo(name, qualifiers, interface);
45:
46: if (Result != NULL)
47:     Result->Join(m_Requests.RegInfo(name, qualifiers, interface));
48: else
49:     Result = m_Requests.RegInfo(name, qualifiers, interface);
50:
51: return Result;
52: //return RegInfo(root,name,qualifiers,interface);
53:
54: }
55:
56: MessageDef* xTEDSItemTree::AllRegInfo(const xTEDSQualifierList& qualifiers, const char*
interface) const
57: {
58: return ExactRegInfo(NULL, qualifiers, interface);
59: }
60:
61: MessageDef* xTEDSItemTree::RegexRegInfo(const char* pattern, const xTEDSQualifierList&
qualifiers, const char* interface) const
62: {
63: MessageDef* Result = NULL;
64:
65: Result = m_Commands.RegexRegInfo(pattern, qualifiers, interface);
66:
67: if (Result != NULL)
68:     Result->Join(m_Notifications.RegexRegInfo(pattern, qualifiers, interface));
69: else
70:     Result = m_Notifications.RegexRegInfo(pattern, qualifiers, interface);
71:
72: if (Result != NULL)
73:     Result->Join(m_Requests.RegexRegInfo(pattern, qualifiers, interface));
74: else
75:     Result = m_Requests.RegexRegInfo(pattern, qualifiers, interface);
76:

```

```

77: return Result;
78: }
79:
80: void xTEDSItemTree::AddVariable(const variable* pNewVariable)
81: {
82: xTEDSVariable* NewVariable = new xTEDSVariable();
83: NewVariable->SetVariable(pNewVariable);
84:
85: m_Variables.addItem(NewVariable);
86: }
87:
88: bool xTEDSItemTree::AddCommand(const command* Command)
89: {
90: if (Command == NULL)
91:     return false;
92:
93: xTEDSCommand* NewCommand = new xTEDSCommand();
94: if (!NewCommand->SetCommand(Command, m_Variables))
95: {
96:     delete NewCommand;
97:     return false;
98: }
99:
100: m_Commands.AddItem(NewCommand);
101: return true;
102: }
103:
104: bool xTEDSItemTree::AddNotification(const notification* Notification)
105: {
106:     if (Notification == NULL)
107:         return false;
108:
109: xTEDSNotification* NewNotification = new xTEDSNotification();
110: if (!NewNotification->SetNotification(Notification, m_Variables))
111: {
112:     delete NewNotification;
113:     return false;
114: }
115:
116: m_Notifications.AddItem(NewNotification);
117: return true;

```

```

118: }
119:
120: bool xTEDSItemTree::AddRequest(const request* Request)
121: {
122:     if (Request == NULL)
123:         return false;
124:
125:     xTEDSRequest* NewRequest = new xTEDSRequest();
126:     if (!NewRequest->SetRequest(Request, m_Variables))
127:     {
128:         delete NewRequest;
129:         return false;
130:     }
131:
132:     m_Requests.AddItem(NewRequest);
133:     return true;
134: }
135:
136: VariableDef* xTEDSItemTree::VarInfoRequest(const char* VarName, const SDMMMessage_ID&
Interface) const
137: {
138:     return m_Variables.VarInfoRequest(VarName, Interface);
139: }
140:
141: SDMMMessage_ID      xTEDSItemTree::GetNotificationFaultID(const      SDMMMessage_ID&
DataMessageID) const
142: {
143:     return m_Notifications.GetFaultID(DataMessageID);
144: }
145:
146: SDMMMessage_ID      xTEDSItemTree::GetCommandFaultID(const      SDMMMessage_ID&
CommandMessageID) const
147: {
148:     return m_Commands.GetFaultID(CommandMessageID);
149: }
150:
151: SDMMMessage_ID      xTEDSItemTree::GetServiceFaultID(const      SDMMMessage_ID&
CommandMessageID) const
152: {
153:     return m_Requests.GetFaultID(CommandMessageID);
154: }
155:

```

```

156:     SDMMMessage_ID      xTEDSItemTree::GetServiceDataID(const    SDMMMessage_ID&
CommandMessageID) const
157: {
158:     return m_Requests.GetDataID(CommandMessageID);
159: }
160:
161: bool xTEDSItemTree::IsServiceIdValid(const SDMMMessage_ID& RequestedId) const
162: {
163:     if (GetServiceDataID (RequestedId) == SDMMMessage_ID(' \0', ' \0'))
164:         return false;
165:     return true;
166: }
167:
168: #ifndef REMOVE_DEBUG_OUTPUT
169: void xTEDSItemTree::PrintDebug()
170: {
171:     m_Variables.PrintDebug();
172:     printf(" \n");
173:     m_Notifications.PrintDebug();
174:     printf(" \n");
175:     m_Commands.PrintDebug();
176:     printf(" \n");
177:     m_Requests.PrintDebug();
178:     printf(" \n");
179: }
180: #endif
181:
182:
183:

```

File: sdm/common/xTEDS/xTEDSCommand.h

```
1: #ifndef __SDM_XTEDS_COMMAND_H_
2: #define __SDM_XTEDS_COMMAND_H_
3:
4: #include "xTEDSCommandMsg.h"
5: #include "xTEDSFaultMsg.h"
6: #include "xTEDSWrapper.h"
7: #include "xTEDSVariableList.h"
8:
9: class xTEDSCommand:public xTEDSWrapper
10: {
11: public:
12: xTEDSCommand();
13: xTEDSCommand(const xTEDSCommand&);
14: virtual ~xTEDSCommand();
15:
16: virtual MessageDef* RegInfo(const char* ItemName) const;
17: virtual bool RegInfoMatch(const char* Name, const xTEDSQualifierList&, const char* Interface)
    const;
18: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
    Interface) const;
19: xTEDSCommand& operator=(const xTEDSCommand&);
20: bool SetCommand(const command* Command, const xTEDSVariableList& VariableList);
21: bool ContainsMessage(const SDMMMessage_ID& MessageID) const;
22: SDMMMessage_ID GetFaultMessageID() const;
23: virtual void PrintDebug() const;
24: private:
25: xTEDSCommandMsg* m_CommandMessage;
26: xTEDSFaultMsg* m_FaultMessage;
27: };
28:
29: #endif
```


File: sdm/common/xTEDS/xTEDS.tab.c

```
1: /* A Bison parser, made by GNU Bison 1.875d. */
2:
3: /* Skeleton parser for Yacc-like parsing with Bison,
4:  Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.
5:
6:  This program is free software; you can redistribute it and/or modify
7:  it under the terms of the GNU General Public License as published by
8:  the Free Software Foundation; either version 2, or (at your option)
9:  any later version.
10:
11:  This program is distributed in the hope that it will be useful,
12:  but WITHOUT ANY WARRANTY; without even the implied warranty of
13:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14:  GNU General Public License for more details.
15:
16:  You should have received a copy of the GNU General Public License
17:  along with this program; if not, write to the Free Software
18:  Foundation, Inc., 59 Temple Place - Suite 330,
19:  Boston, MA 02111-1307, USA. */
20:
21: /* As a special exception, when this file is copied by Bison into a
22:  Bison output file, you may use that output file without restriction.
23:  This special exception was added by the Free Software Foundation
24:  in version 1.24 of Bison. */
25:
26: /* Written by Richard Stallman by simplifying the original so called
27:  ``semantic" parser. */
28:
29: /* All symbols defined below should begin with yy or YY, to avoid
30:  infringing on user name space.  This should be done even for local
31:  variables, as they might otherwise be expanded by user macros.
32:  There are some unavoidable exceptions within include files to
33:  define necessary library symbols; they are noted "INFRINGES ON
34:  USER NAME SPACE" below. */
35:
36: /* Identify Bison output. */
37: #define YYBISON 1
38:
39: /* Skeleton name. */
```

```

40: #define YYSKELETON_NAME "yacc.c"
41:
42: /* Pure parsers. */
43: #define YYPURE 0
44:
45: /* Using locations. */
46: #define YYLSP_NEEDED 0
47:
48: /* If NAME_PREFIX is specified substitute the variables and functions
49:  names. */
50: #define yyparse xTEDSparsed
51: #define yylex xTEDSlex
52: #define yyerror xTEDSerror
53: #define yylval xTEDSlval
54: #define yychar xTEDSchar
55: #define yydebug xTEDSdebug
56: #define yynerrs xTEDSnerrs
57:
58:
59: /* Tokens. */
60: #ifndef YYTOKENTYPE
61: # define YYTOKENTYPE
62:  /* Put the tokens into the symbol table, so that GDB and other debuggers
63:   know about them. */
64:  enum yytokentype {
65:   EQUAL_SY = 258,
66:   CLOSE_SY = 259,
67:   SLASHCLOSE_SY = 260,
68:   OPEN_XML_SY = 261,
69:   CLOSE_xTEDS_SY = 262,
70:   OPEN_xTEDS_SY = 263,
71:   OPEN_APP_SY = 264,
72:   OPEN_VAR_SY = 265,
73:   CLOSE_VAR_SY = 266,
74:   OPEN_DRANGE_SY = 267,
75:   CLOSE_DRANGE_SY = 268,
76:   OPEN_OPTION_SY = 269,
77:   OPEN_CURVE_SY = 270,
78:   CLOSE_CURVE_SY = 271,
79:   OPEN_COEFF_SY = 272,
80:   OPEN_DATA_MSG_SY = 273,

```

81: CLOSE_DATA_MSG_SY = 274,
82: OPEN_VARIABLE_REF_SY = 275,
83: OPEN_COMMAND_MSG_SY = 276,
84: CLOSE_COMMAND_MSG_SY = 277,
85: NAME_SY = 278,
86: KIND_SY = 279,
87: ID_SY = 280,
88: CLOSE_ORIENTATION_SY = 281,
89: QUALIFIER_SY = 282,
90: DESCRIPTION_SY = 283,
91: MANUFACTURER_ID_SY = 284,
92: VERSION_SY = 285,
93: MODEL_ID_SY = 286,
94: VERSION_LETTER_SY = 287,
95: SERIAL_NUMBER_SY = 288,
96: CALIBRATION_DATE_SY = 289,
97: SENSITIVITY_AT_REF_SY = 290,
98: REF_FREQ_SY = 291,
99: REF_TEMP_SY = 292,
100: MEASUREMENT_RANGE_SY = 293,
101: ELECTRICAL_OUTPUT_SY = 294,
102: QUALITY_FACTOR_SY = 295,
103: TEMP_COEFF_SY = 296,
104: DIRECTION_XYZ_SY = 297,
105: CAL_DUE_DATE_SY = 298,
106: POWER_REQS_SY = 299,
107: VALUE_SY = 300,
108: ALARM_SY = 301,
109: MSG_ARRIVAL_SY = 302,
110: MSG_RATE_SY = 303,
111: STRING = 304,
112: FLOAT = 305,
113: INT = 306,
114: PRECISION_SY = 307,
115: RANGE_MAX_SY = 308,
116: CLOSE_LOCATION_SY = 309,
117: FORMAT_SY = 310,
118: ACCURACY_SY = 311,
119: RANGE_MIN_SY = 312,
120: SCALE_FACTOR_SY = 313,
121: UNITS_SY = 314,

122: DEFAULT_VALUE_SY = 315,
123: OPEN_DEVICE_SY = 316,
124: SCALE_UNITS_SY = 317,
125: LENGTH_SY = 318,
126: EXPONENT_SY = 319,
127: SCHEMA_LOCATION_SY = 320,
128: XMLNS_SY = 321,
129: XMLNS_XSI_SY = 322,
130: CLOSE_OPTION_SY = 323,
131: OPEN_INTERFACE_SY = 324,
132: OPEN_COMMAND_SY = 325,
133: OPEN_NOTIFICATION_SY = 326,
134: OPEN_REQUEST_SY = 327,
135: OPEN_FAULT_MSG_SY = 328,
136: COMPONENT_KEY_SY = 329,
137: SPA_U_HUB_SY = 330,
138: SPA_U_PORT_SY = 331,
139: EXTENDS_SY = 332,
140: CLOSE_COMMAND_SY = 333,
141: CLOSE_NOTIFICATION_SY = 334,
142: CLOSE_REQUEST_SY = 335,
143: CLOSE_FAULT_MSG_SY = 336,
144: OPEN_QUALIFIER_SY = 337,
145: CLOSE_QUALIFIER_SY = 338,
146: CLOSE_APP_SY = 339,
147: CLOSE_DEVICE_SY = 340,
148: CLOSE_INTERFACE_SY = 341,
149: MEMORY_MINIMUM_SY = 342,
150: OPERATING_SYSTEM_SY = 343,
151: PATH_FOR_ASSEMBLY_SY = 344,
152: PATH_ON_SPACECRAFT_SY = 345,
153: X_SY = 346,
154: Y_SY = 347,
155: Z_SY = 348,
156: AXIS_SY = 349,
157: ANGLE_SY = 350,
158: OPEN_LOCATION_SY = 351,
159: OPEN_ORIENTATION_SY = 352,
160: CLOSE_XML_SY = 353,
161: ENCODING_SY = 354,
162: STANDALONE_SY = 355,

```

163:  CLOSE_VARIABLE_REF_SY = 356,
164:  CLOSE_COEFF_SY = 357,
165:  R_LOW_SY = 358,
166:  R_HIGH_SY = 359,
167:  Y_LOW_SY = 360,
168:  Y_HIGH_SY = 361,
169:  INVALID_VALUE_SY = 362,
170:  BAD_TERMINAL_SY = 363
171:  };
172: #endif
173: #define EQUAL_SY 258
174: #define CLOSE_SY 259
175: #define SLASHCLOSE_SY 260
176: #define OPEN_XML_SY 261
177: #define CLOSE_xTEDS_SY 262
178: #define OPEN_xTEDS_SY 263
179: #define OPEN_APP_SY 264
180: #define OPEN_VAR_SY 265
181: #define CLOSE_VAR_SY 266
182: #define OPEN_DRANGE_SY 267
183: #define CLOSE_DRANGE_SY 268
184: #define OPEN_OPTION_SY 269
185: #define OPEN_CURVE_SY 270
186: #define CLOSE_CURVE_SY 271
187: #define OPEN_COEFF_SY 272
188: #define OPEN_DATA_MSG_SY 273
189: #define CLOSE_DATA_MSG_SY 274
190: #define OPEN_VARIABLE_REF_SY 275
191: #define OPEN_COMMAND_MSG_SY 276
192: #define CLOSE_COMMAND_MSG_SY 277
193: #define NAME_SY 278
194: #define KIND_SY 279
195: #define ID_SY 280
196: #define CLOSE_ORIENTATION_SY 281
197: #define QUALIFIER_SY 282
198: #define DESCRIPTION_SY 283
199: #define MANUFACTURER_ID_SY 284
200: #define VERSION_SY 285
201: #define MODEL_ID_SY 286
202: #define VERSION_LETTER_SY 287
203: #define SERIAL_NUMBER_SY 288

```

204: #define CALIBRATION_DATE_SY 289
205: #define SENSITIVITY_AT_REF_SY 290
206: #define REF_FREQ_SY 291
207: #define REF_TEMP_SY 292
208: #define MEASUREMENT_RANGE_SY 293
209: #define ELECTRICAL_OUTPUT_SY 294
210: #define QUALITY_FACTOR_SY 295
211: #define TEMP_COEFF_SY 296
212: #define DIRECTION_XYZ_SY 297
213: #define CAL_DUE_DATE_SY 298
214: #define POWER_REQS_SY 299
215: #define VALUE_SY 300
216: #define ALARM_SY 301
217: #define MSG_ARRIVAL_SY 302
218: #define MSG_RATE_SY 303
219: #define STRING 304
220: #define FLOAT 305
221: #define INT 306
222: #define PRECISION_SY 307
223: #define RANGE_MAX_SY 308
224: #define CLOSE_LOCATION_SY 309
225: #define FORMAT_SY 310
226: #define ACCURACY_SY 311
227: #define RANGE_MIN_SY 312
228: #define SCALE_FACTOR_SY 313
229: #define UNITS_SY 314
230: #define DEFAULT_VALUE_SY 315
231: #define OPEN_DEVICE_SY 316
232: #define SCALE_UNITS_SY 317
233: #define LENGTH_SY 318
234: #define EXPONENT_SY 319
235: #define SCHEMA_LOCATION_SY 320
236: #define XMLNS_SY 321
237: #define XMLNS_XSI_SY 322
238: #define CLOSE_OPTION_SY 323
239: #define OPEN_INTERFACE_SY 324
240: #define OPEN_COMMAND_SY 325
241: #define OPEN_NOTIFICATION_SY 326
242: #define OPEN_REQUEST_SY 327
243: #define OPEN_FAULT_MSG_SY 328
244: #define COMPONENT_KEY_SY 329

```

245: #define SPA_U_HUB_SY 330
246: #define SPA_U_PORT_SY 331
247: #define EXTENDS_SY 332
248: #define CLOSE_COMMAND_SY 333
249: #define CLOSE_NOTIFICATION_SY 334
250: #define CLOSE_REQUEST_SY 335
251: #define CLOSE_FAULT_MSG_SY 336
252: #define OPEN_QUALIFIER_SY 337
253: #define CLOSE_QUALIFIER_SY 338
254: #define CLOSE_APP_SY 339
255: #define CLOSE_DEVICE_SY 340
256: #define CLOSE_INTERFACE_SY 341
257: #define MEMORY_MINIMUM_SY 342
258: #define OPERATING_SYSTEM_SY 343
259: #define PATH_FOR_ASSEMBLY_SY 344
260: #define PATH_ON_SPACECRAFT_SY 345
261: #define X_SY 346
262: #define Y_SY 347
263: #define Z_SY 348
264: #define AXIS_SY 349
265: #define ANGLE_SY 350
266: #define OPEN_LOCATION_SY 351
267: #define OPEN_ORIENTATION_SY 352
268: #define CLOSE_XML_SY 353
269: #define ENCODING_SY 354
270: #define STANDALONE_SY 355
271: #define CLOSE_VARIABLE_REF_SY 356
272: #define CLOSE_COEFF_SY 357
273: #define R_LOW_SY 358
274: #define R_HIGH_SY 359
275: #define Y_LOW_SY 360
276: #define Y_HIGH_SY 361
277: #define INVALID_VALUE_SY 362
278: #define BAD_TERMINAL_SY 363
279:
280:
281:
282:
283: /* Copy the first part of user declarations. */
284: #line 1 "xTEDS.y"
285:

```

```

286: /*xTEDS 1.0 msg_def parser*/
287: #include <stdio.h>
288: #include <stdlib.h>
289: #include <string.h>
290:
291: #include "xTEDSParser.h"
292:
293: int yylex();
294:
295: int yydebug=0;
296: void yyerror(char *s);
297:
298:
299: /* Enabling traces. */
300: #ifndef YYDEBUG
301: # define YYDEBUG 1
302: #endif
303:
304: /* Enabling verbose error messages. */
305: #ifdef YYERROR_VERBOSE
306: # undef YYERROR_VERBOSE
307: # define YYERROR_VERBOSE 1
308: #else
309: # define YYERROR_VERBOSE 0
310: #endif
311:
312: #if ! defined (YYSTYPE) && ! defined (YYSTYPE_IS_DECLARED)
313: #line 38 "xTEDS.y"
314: typedef union YYSTYPE {
315:     int integer;
316:     float real;
317:     char* str;
318:     struct variable_data* var;
319:     struct variable_reference* var_ref;
320:     struct qualifier_data* qual;
321:     struct coefficient_data* coef;
322:     struct curve_data* curve;
323:     struct option_data* curveoption;
324:     struct drange_data* drange;
325:     struct location_data* location;
326:     struct orientation_data* orientation;

```



```

327: struct fault_message* fault_msg;
328: struct data_message* data_msg;
329: struct command_message* cmd_msg;
330: struct command_type* command;
331: struct notification_type* notification;
332: struct request_type* request;
333: struct message_type* message;
334: struct interface_type* interface;
335: struct xteds* xteds;
336: struct app_device_attributes* attr;
337: } YYSTYPE;
338: /* Line 191 of yacc.c. */
339: #line 340 "xTEDS.tab.c"
340: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
341: # define YYSTYPE_IS_DECLARED 1
342: # define YYSTYPE_IS_TRIVIAL 1
343: #endif
344:
345:
346:
347: /* Copy the second part of user declarations. */
348:
349:
350: /* Line 214 of yacc.c. */
351: #line 352 "xTEDS.tab.c"
352:
353: #if ! defined (yyoverflow) || YYERROR_VERBOSE
354:
355: # ifndef YYFREE
356: #  define YYFREE free
357: # endif
358: # ifndef YYMALLOC
359: #  define YYMALLOC malloc
360: # endif
361:
362: /* The parser invokes alloca or malloc; define the necessary symbols. */
363:
364: # ifdef YYSTACK_USE_ALLOCA
365: #  if YYSTACK_USE_ALLOCA
366: #   define YYSTACK_ALLOC alloca
367: #  endif

```

```

368: # else
369: # if defined (alloca) || defined (_ALLOCA_H)
370: #   define YYSTACK_ALLOC alloca
371: # else
372: #   ifdef __GNUC__
373: #     define YYSTACK_ALLOC __builtin_alloca
374: #   endif
375: # endif
376: # endif
377:
378: # ifdef YYSTACK_ALLOC
379: /* Pacify GCC's 'empty if-body' warning. */
380: # define YYSTACK_FREE(Ptr) do { /* empty */; } while (0)
381: # else
382: # if defined (__STDC__) || defined (__cplusplus)
383: #   include <stdlib.h> /* INFRINGES ON USER NAME SPACE */
384: #   define YYSIZE_T size_t
385: # endif
386: # define YYSTACK_ALLOC YMALLOC
387: # define YYSTACK_FREE YYFREE
388: # endif
389: #endif /* ! defined (yyoverflow) || YYERROR_VERBOSE */
390:
391:
392: #if (! defined (yyoverflow) \
393:    && (! defined (__cplusplus) \
394:    || (defined (YYSTYPE_IS_TRIVIAL) && YYSTYPE_IS_TRIVIAL))))
395:
396: /* A type that is properly aligned for any stack member. */
397: union yyallocc
398: {
399:   short int yyss;
400:   YYSTYPE yyvs;
401: };
402:
403: /* The size of the maximum gap between one aligned stack and the next. */
404: # define YYSTACK_GAP_MAXIMUM (sizeof (union yyallocc) - 1)
405:
406: /* The size of an array large to enough to hold all stacks, each with
407:   N elements. */
408: # define YYSTACK_BYTES(N) \

```

```

409: ((N) * (sizeof (short int) + sizeof (YYSTYPE))          \
410:   + YYSTACK_GAP_MAXIMUM)
411:
412: /* Copy COUNT objects from FROM to TO. The source and destination do
413:  not overlap. */
414: # ifndef YYCOPY
415: #  if defined (__GNUC__) && 1 < __GNUC__
416: #   define YYCOPY(To, From, Count) \
417:     __builtin_memcpy (To, From, (Count) * sizeof (*(From)))
418: #  else
419: #   define YYCOPY(To, From, Count) \
420:     do \
421:     { \
422:       register YYSIZE_T yyi; \
423:       for (yyi = 0; yyi < (Count); yyi++) \
424:         (To)[yyi] = (From)[yyi]; \
425:     } \
426:     while (0)
427: #  endif
428: # endif
429:
430: /* Relocate STACK from its old location to the new one. The
431:  local variables YYSIZE and YYSTACKSIZE give the old and new number of
432:  elements in the stack, and YYPTR gives the new location of the
433:  stack. Advance YYPTR to a properly aligned location for the next
434:  stack. */
435: # define YYSTACK_RELOCATE(Stack) \
436:  do \
437:  { \
438:    YYSIZE_T yynewbytes; \
439:    YYCOPY (&yyptr->Stack, Stack, yysize); \
440:    Stack = &yyptr->Stack; \
441:    yynewbytes = yystacksize * sizeof (*Stack) + YYSTACK_GAP_MAXIMUM; \
442:    yyptr += yynewbytes / sizeof (*yyptr); \
443:  } \
444:  while (0)
445:
446: #endif
447:
448: #if defined (__STDC__) || defined (__cplusplus)
449: typedef signed char yysigned_char;

```

```

450: #else
451:  typedef short int yysigned_char;
452: #endif
453:
454: /* YYFINAL -- State number of the termination state. */
455: #define YYFINAL  4
456: /* YYLAST -- Last index in YYTABLE. */
457: #define YYLAST  428
458:
459: /* YYNTOKENS -- Number of terminals. */
460: #define YYNTOKENS  109
461: /* YYNNTS -- Number of nonterminals. */
462: #define YYNNTS  84
463: /* YYNRULES -- Number of rules. */
464: #define YYNRULES  215
465: /* YYNRULES -- Number of states. */
466: #define YYNSTATES  478
467:
468: /* YYTRANSLATE(YYLEX) -- Bison symbol number corresponding to YYLEX. */
469: #define YYUNDEFTOK  2
470: #define YYMAXUTOK  363
471:
472: #define YYTRANSLATE(YYX)                                     \
473:  ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)
474:
475: /* YYTRANSLATE[YYLEX] -- Bison symbol number corresponding to YYLEX. */
476: static const unsigned char yytranslate[] =
477: {
478:   0,  2,  2,  2,  2,  2,  2,  2,  2,  2,
479:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
480:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
481:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
482:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
483:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
484:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
485:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
486:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
487:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
488:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
489:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
490:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,

```

```

491:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
492:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
493:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
494:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
495:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
496:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
497:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
498:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
499:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
500:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
501:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
502:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
503:  2,  2,  2,  2,  2,  2,  1,  2,  3,  4,
504:  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
505: 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
506: 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
507: 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
508: 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
509: 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
510: 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
511: 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
512: 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
513: 95, 96, 97, 98, 99, 100, 101, 102, 103, 104,
514: 105, 106, 107, 108
515: };
516:
517: #if YYDEBUG
518: /* YYPRHS[YYN] -- Index of the first RHS symbol of rule number YYN in
519:  YYRHS. */
520: static const unsigned short int yyprhs[] =
521: {
522:  0,  0,  3, 12, 16, 17, 21, 22, 28, 32,
523: 35, 36, 40, 44, 48, 52, 56, 60, 62, 64,
524: 68, 75, 78, 79, 83, 87, 91, 95, 99, 103,
525: 107, 109, 113, 117, 121, 125, 129, 133, 141, 144,
526: 145, 147, 149, 151, 154, 155, 157, 161, 165, 169,
527: 173, 177, 181, 185, 189, 193, 197, 201, 205, 209,
528: 213, 217, 221, 225, 229, 235, 238, 239, 243, 247,
529: 251, 255, 259, 265, 268, 269, 273, 277, 281, 284,
530: 285, 291, 295, 298, 299, 303, 307, 311, 314, 315,
531: 322, 324, 328, 332, 335, 336, 339, 340, 342, 344,

```

```

532: 346, 350, 354, 358, 362, 365, 366, 368, 370, 372,
533: 380, 382, 390, 394, 395, 398, 399, 403, 407, 411,
534: 413, 420, 423, 424, 432, 435, 436, 440, 444, 448,
535: 452, 456, 458, 465, 468, 469, 477, 481, 484, 485,
536: 489, 493, 497, 499, 502, 503, 505, 507, 513, 516,
537: 519, 522, 523, 527, 531, 535, 539, 543, 547, 551,
538: 555, 559, 563, 567, 571, 575, 579, 583, 587, 591,
539: 595, 599, 603, 606, 608, 611, 612, 614, 616, 618,
540: 620, 623, 624, 630, 638, 643, 647, 650, 651, 655,
541: 659, 662, 663, 667, 673, 676, 677, 681, 685, 689,
542: 693, 698, 702, 705, 706, 710, 714, 717, 718, 722,
543: 728, 732, 735, 736, 740, 744
544: };
545:
546: /* YYRHS -- A '-1'-separated list of the rules' RHS. */
547: static const short int yyrhs[] =
548: {
549: 110, 0, -1, 6, 30, 3, 49, 111, 112, 98,
550: 113, -1, 99, 3, 49, -1, -1, 100, 3, 49,
551: -1, -1, 114, 117, 137, 7, 4, -1, 8, 115,
552: 4, -1, 115, 116, -1, -1, 23, 3, 49, -1,
553: 30, 3, 49, -1, 28, 3, 49, -1, 66, 3,
554: 49, -1, 65, 3, 49, -1, 67, 3, 49, -1,
555: 118, -1, 122, -1, 9, 119, 5, -1, 9, 119,
556: 4, 133, 84, 4, -1, 119, 121, -1, -1, 23,
557: 3, 49, -1, 24, 3, 49, -1, 25, 3, 49,
558: -1, 27, 3, 49, -1, 28, 3, 49, -1, 29,
559: 3, 49, -1, 74, 3, 49, -1, 120, -1, 30,
560: 3, 49, -1, 87, 3, 49, -1, 88, 3, 49,
561: -1, 89, 3, 49, -1, 90, 3, 49, -1, 61,
562: 125, 5, -1, 61, 125, 4, 123, 85, 4, 133,
563: -1, 123, 124, -1, -1, 134, -1, 127, -1, 130,
564: -1, 125, 126, -1, -1, 120, -1, 31, 3, 49,
565: -1, 32, 3, 49, -1, 33, 3, 49, -1, 34,
566: 3, 49, -1, 35, 3, 49, -1, 36, 3, 49,
567: -1, 37, 3, 49, -1, 38, 3, 49, -1, 39,
568: 3, 49, -1, 40, 3, 49, -1, 41, 3, 49,
569: -1, 42, 3, 49, -1, 43, 3, 49, -1, 44,
570: 3, 49, -1, 75, 3, 49, -1, 76, 3, 49,
571: -1, 30, 3, 49, -1, 96, 128, 5, -1, 96,
572: 128, 4, 54, 4, -1, 128, 129, -1, -1, 91,

```

573: 3, 49, -1, 92, 3, 49, -1, 93, 3, 49,
 574: -1, 59, 3, 49, -1, 97, 131, 5, -1, 97,
 575: 131, 4, 26, 4, -1, 131, 132, -1, -1, 94,
 576: 3, 49, -1, 95, 3, 49, -1, 59, 3, 49,
 577: -1, 133, 134, -1, -1, 82, 135, 4, 83, 4,
 578: -1, 82, 135, 5, -1, 135, 136, -1, -1, 23,
 579: 3, 49, -1, 45, 3, 49, -1, 59, 3, 49,
 580: -1, 137, 138, -1, -1, 139, 141, 164, 144, 86,
 581: 4, -1, 139, -1, 69, 140, 4, -1, 69, 140,
 582: 5, -1, 140, 143, -1, -1, 141, 142, -1, -1,
 583: 134, -1, 127, -1, 130, -1, 23, 3, 49, -1,
 584: 77, 3, 49, -1, 25, 3, 49, -1, 28, 3,
 585: 49, -1, 144, 145, -1, -1, 158, -1, 152, -1,
 586: 146, -1, 72, 4, 159, 153, 147, 80, 4, -1,
 587: 148, -1, 73, 149, 4, 151, 175, 81, 4, -1,
 588: 73, 149, 5, -1, -1, 149, 150, -1, -1, 23,
 589: 3, 49, -1, 25, 3, 49, -1, 28, 3, 49,
 590: -1, 133, -1, 71, 4, 153, 147, 79, 4, -1,
 591: 153, 154, -1, -1, 18, 155, 4, 157, 175, 19,
 592: 4, -1, 155, 156, -1, -1, 23, 3, 49, -1,
 593: 25, 3, 49, -1, 47, 3, 49, -1, 28, 3,
 594: 49, -1, 48, 3, 49, -1, 133, -1, 70, 4,
 595: 159, 147, 78, 4, -1, 159, 160, -1, -1, 21,
 596: 161, 4, 163, 175, 22, 4, -1, 21, 161, 5,
 597: -1, 161, 162, -1, -1, 23, 3, 49, -1, 25,
 598: 3, 49, -1, 28, 3, 49, -1, 133, -1, 164,
 599: 165, -1, -1, 166, -1, 167, -1, 168, 4, 171,
 600: 11, 4, -1, 168, 5, -1, 10, 169, -1, 169,
 601: 170, -1, -1, 23, 3, 49, -1, 24, 3, 49,
 602: -1, 55, 3, 49, -1, 27, 3, 49, -1, 25,
 603: 3, 49, -1, 28, 3, 49, -1, 57, 3, 49,
 604: -1, 53, 3, 49, -1, 63, 3, 49, -1, 60,
 605: 3, 49, -1, 52, 3, 49, -1, 59, 3, 49,
 606: -1, 56, 3, 49, -1, 58, 3, 49, -1, 62,
 607: 3, 49, -1, 103, 3, 49, -1, 104, 3, 49,
 608: -1, 105, 3, 49, -1, 106, 3, 49, -1, 107,
 609: 3, 49, -1, 172, 173, -1, 133, -1, 173, 174,
 610: -1, -1, 177, -1, 185, -1, 127, -1, 130, -1,
 611: 175, 176, -1, -1, 20, 23, 3, 49, 5, -1,
 612: 20, 23, 3, 49, 4, 101, 4, -1, 178, 181,
 613: 13, 4, -1, 12, 179, 4, -1, 179, 180, -1,

```

614:  -1, 23, 3, 49, -1, 28, 3, 49, -1, 181,
615: 182, -1, -1, 14, 183, 5, -1, 14, 183, 4,
616: 68, 4, -1, 183, 184, -1, -1, 23, 3, 49,
617: -1, 45, 3, 49, -1, 28, 3, 49, -1, 46,
618: 3, 49, -1, 186, 189, 16, 4, -1, 15, 187,
619: 4, -1, 187, 188, -1, -1, 23, 3, 49, -1,
620: 28, 3, 49, -1, 189, 190, -1, -1, 17, 191,
621: 5, -1, 17, 191, 4, 102, 4, -1, 17, 191,
622: 4, -1, 191, 192, -1, -1, 64, 3, 49, -1,
623: 45, 3, 49, -1, 28, 3, 49, -1
624: };
625:
626: /* YYRLINE[YYN] -- source line where rule number YYN was defined. */
627: static const unsigned short int yyrline[] =
628: {
629: 0, 92, 92, 106, 116, 121, 125, 133, 139, 145,
630: 150, 157, 164, 171, 178, 185, 192, 206, 210, 216,
631: 220, 229, 234, 239, 246, 253, 260, 268, 275, 282,
632: 291, 295, 302, 309, 316, 323, 337, 341, 351, 356,
633: 361, 367, 372, 379, 384, 389, 393, 400, 407, 414,
634: 421, 428, 435, 442, 449, 456, 463, 470, 477, 484,
635: 491, 498, 505, 519, 523, 529, 534, 539, 545, 551,
636: 557, 570, 574, 580, 585, 590, 596, 602, 614, 619,
637: 624, 628, 634, 639, 644, 651, 658, 672, 677, 682,
638: 686, 692, 696, 702, 707, 714, 720, 725, 729, 735,
639: 743, 750, 757, 764, 778, 783, 788, 795, 802, 811,
640: 826, 832, 838, 843, 848, 853, 858, 865, 872, 881,
641: 895, 904, 909, 914, 922, 927, 932, 939, 946, 953,
642: 960, 969, 983, 992, 997, 1002, 1008, 1014, 1019, 1024,
643: 1031, 1038, 1047, 1061, 1066, 1071, 1075, 1081, 1087, 1090,
644: 1096, 1101, 1106, 1113, 1120, 1127, 1135, 1142, 1149, 1156,
645: 1163, 1170, 1177, 1184, 1191, 1198, 1205, 1212, 1219, 1226,
646: 1233, 1241, 1250, 1256, 1266, 1271, 1276, 1283, 1290, 1297,
647: 1311, 1316, 1321, 1325, 1336, 1345, 1351, 1356, 1361, 1368,
648: 1377, 1382, 1387, 1391, 1397, 1402, 1407, 1414, 1421, 1428,
649: 1442, 1451, 1457, 1462, 1467, 1474, 1483, 1488, 1493, 1497,
650: 1501, 1507, 1512, 1517, 1524, 1531
651: };
652: #endif
653:
654: #if YYDEBUG || YYERROR_VERBOSE

```



```

655: /* YYTNME[SYMBOL-NUM] -- String name of the symbol SYMBOL-NUM.
656: First, the terminals, then, starting at YYNTOKENS, nonterminals. */
657: static const char *const yytnme[] =
658: {
659: "$end", "error", "$undefined", "EQUAL_SY", "CLOSE_SY", "SLASHCLOSE_SY",
660: "OPEN_XML_SY", "CLOSE_xTEDS_SY", "OPEN_xTEDS_SY", "OPEN_APP_SY",
661: "OPEN_VAR_SY", "CLOSE_VAR_SY", "OPEN_DRANGE_SY", "CLOSE_DRANGE_SY",
662: "OPEN_OPTION_SY", "OPEN_CURVE_SY", "CLOSE_CURVE_SY", "OPEN_COEFF_SY",
663: "OPEN_DATA_MSG_SY", "CLOSE_DATA_MSG_SY", "OPEN_VARIABLE_REF_SY",
664: "OPEN_COMMAND_MSG_SY", "CLOSE_COMMAND_MSG_SY", "NAME_SY",
"KIND_SY",
665: "ID_SY", "CLOSE_ORIENTATION_SY", "QUALIFIER_SY", "DESCRIPTION_SY",
666: "MANUFACTURER_ID_SY", "VERSION_SY", "MODEL_ID_SY", "VERSION_LETTER_SY",
667: "SERIAL_NUMBER_SY", "CALIBRATION_DATE_SY", "SENSITIVITY_AT_REF_SY",
668: "REF_FREQ_SY", "REF_TEMP_SY", "MEASUREMENT_RANGE_SY",
669: "ELECTRICAL_OUTPUT_SY", "QUALITY_FACTOR_SY", "TEMP_COEFF_SY",
670: "DIRECTION_XYZ_SY", "CAL_DUE_DATE_SY", "POWER_REQS_SY", "VALUE_SY",
671: "ALARM_SY", "MSG_ARRIVAL_SY", "MSG_RATE_SY", "STRING", "FLOAT", "INT",
672: "PRECISION_SY", "RANGE_MAX_SY", "CLOSE_LOCATION_SY", "FORMAT_SY",
673: "ACCURACY_SY", "RANGE_MIN_SY", "SCALE_FACTOR_SY", "UNITS_SY",
674: "DEFAULT_VALUE_SY", "OPEN_DEVICE_SY", "SCALE_UNITS_SY", "LENGTH_SY",
675: "EXPONENT_SY", "SCHEMA_LOCATION_SY", "XMLNS_SY", "XMLNS_XSI_SY",
676: "CLOSE_OPTION_SY", "OPEN_INTERFACE_SY", "OPEN_COMMAND_SY",
677: "OPEN_NOTIFICATION_SY", "OPEN_REQUEST_SY", "OPEN_FAULT_MSG_SY",
678: "COMPONENT_KEY_SY", "SPA_U_HUB_SY", "SPA_U_PORT_SY", "EXTENDS_SY",
679: "CLOSE_COMMAND_SY", "CLOSE_NOTIFICATION_SY", "CLOSE_REQUEST_SY",
680: "CLOSE_FAULT_MSG_SY", "OPEN_QUALIFIER_SY", "CLOSE_QUALIFIER_SY",
681: "CLOSE_APP_SY", "CLOSE_DEVICE_SY", "CLOSE_INTERFACE_SY",
682: "MEMORY_MINIMUM_SY", "OPERATING_SYSTEM_SY", "PATH_FOR_ASSEMBLY_SY",
683: "PATH_ON_SPACECRAFT_SY", "X_SY", "Y_SY", "Z_SY", "AXIS_SY", "ANGLE_SY",
684: "OPEN_LOCATION_SY", "OPEN_ORIENTATION_SY", "CLOSE_XML_SY",
"ENCODING_SY",
685: "STANDALONE_SY", "CLOSE_VARIABLE_REF_SY", "CLOSE_COEFF_SY",
"R_LOW_SY",
686: "R_HIGH_SY", "Y_LOW_SY", "Y_HIGH_SY", "INVALID_VALUE_SY",
687: "BAD_TERMINAL_SY", "$accept", "XTEDS_DOCUMENT", "ENCODING",
"STANDALONE",
688: "xTEDS", "OPEN_xTEDS", "xTEDS_ATTRIBUTES", "xTEDS_ATTRIBUTE",
689: "APP_DEVICE", "APP_SECTION", "APP_ATTRIBUTES",
690: "COMMON_APP_DEVICE_ATTRIBUTE", "APP_ATTRIBUTE", "DEVICE_SECTION",
691: "DEVICE_SUBELEMENTS", "DEVICE_SUBELEMENT", "DEVICE_ATTRIBUTES",
692: "DEVICE_ATTRIBUTE", "LOCATION_SECTION", "LOCATION_ATTRIBUTES",

```

```

693: "LOCATION_ATTRIBUTE", "ORIENTATION_SECTION", "ORIENTATION_ATTRIBUTES",
694: "ORIENTATION_ATTRIBUTE", "QUALIFIERS_SECTION", "QUALIFIER",
695: "QUALIFIERS_ATTRIBUTES", "QUALIFIERS_ATTRIBUTE", "INTERFACES",
696: "INTERFACE", "INTERFACE_HEAD", "INTERFACE_ATTRIBUTES",
697:         "INTERFACE_SUBELEMENTS",         "INTERFACE_SUBELEMENT",
"INTERFACE_ATTRIBUTE",
698: "MESSAGE_SECTION", "MESSAGES", "REQUEST_SECTION", "FAULT_MSG_SECTION",
699: "FAULT_MSG", "FAULT_MSG_ATTRIBUTES", "FAULT_MSG_ATTRIBUTE",
700: "FAULT_MSG_QUALIFIERS", "NOTIFICATION_SECTION", "DATA_MSG_SECTION",
701: "DATA_MSG", "DATA_MSG_ATTRIBUTES", "DATA_MSG_ATTRIBUTE",
702: "DATA_MSG_QUALIFIERS", "COMMAND_SECTION", "COMMAND_MSG_SECTION",
703:         "COMMAND_MSG",         "COMMAND_MSG_ATTRIBUTES",
"COMMAND_MSG_ATTRIBUTE",
704: "COMMAND_MSG_QUALIFIERS", "VAR_SECTION", "VARIABLE",
705: "VAR_WITH_SUBELEMENTS", "VAR_NO_SUBELEMENTS", "VAR_HEAD",
706: "VAR_ATTRIBUTES", "VAR_ATTRIBUTE", "VAR_ELEMENTS", "VAR_QUALIFIERS",
707: "VAR_SUBELEMENTS", "VAR_SUBELEMENT", "VARIABLE_REFS", "VARIABLE_REF",
708: "DRANGE", "DRANGE_HEAD", "DRANGE_ATTRIBUTES", "DRANGE_ATTRIBUTE",
709: "DRANGE_OPTIONS", "DRANGE_OPTION", "OPTION_ATTRIBUTES",
710: "OPTION_ATTRIBUTE", "CURVE", "CURVE_HEAD", "CURVE_ATTRIBUTES",
711: "CURVE_ATTRIBUTE", "CURVE_COEFFS", "CURVE_COEFF", "COEFF_ATTRIBUTES",
712: "COEFF_ATTRIBUTE", 0
713: };
714: #endif
715:
716: # ifdef YYPRINT
717: /* YYTOKNUM[YYLEX-NUM] -- Internal token number corresponding to
718:  token YYLEX-NUM. */
719: static const unsigned short int yytoknum[] =
720: {
721:     0, 256, 257, 258, 259, 260, 261, 262, 263, 264,
722:     265, 266, 267, 268, 269, 270, 271, 272, 273, 274,
723:     275, 276, 277, 278, 279, 280, 281, 282, 283, 284,
724:     285, 286, 287, 288, 289, 290, 291, 292, 293, 294,
725:     295, 296, 297, 298, 299, 300, 301, 302, 303, 304,
726:     305, 306, 307, 308, 309, 310, 311, 312, 313, 314,
727:     315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
728:     325, 326, 327, 328, 329, 330, 331, 332, 333, 334,
729:     335, 336, 337, 338, 339, 340, 341, 342, 343, 344,
730:     345, 346, 347, 348, 349, 350, 351, 352, 353, 354,
731:     355, 356, 357, 358, 359, 360, 361, 362, 363

```

```

732: };
733: #endif
734:
735: /* YYR1[YYN] -- Symbol number of symbol that rule YYN derives. */
736: static const unsigned char yyr1[] =
737: {
738:     0, 109, 110, 111, 111, 112, 112, 113, 114, 115,
739:     115, 116, 116, 116, 116, 116, 116, 117, 117, 118,
740:     118, 119, 119, 120, 120, 120, 120, 120, 120, 120,
741:     121, 121, 121, 121, 121, 121, 122, 122, 123, 123,
742:     124, 124, 124, 125, 125, 126, 126, 126, 126, 126,
743:     126, 126, 126, 126, 126, 126, 126, 126, 126, 126,
744:     126, 126, 126, 127, 127, 128, 128, 129, 129, 129,
745:     129, 130, 130, 131, 131, 132, 132, 132, 133, 133,
746:     134, 134, 135, 135, 136, 136, 136, 137, 137, 138,
747:     138, 139, 139, 140, 140, 141, 141, 142, 142, 142,
748:     143, 143, 143, 143, 144, 144, 145, 145, 145, 146,
749:     147, 148, 148, 148, 149, 149, 150, 150, 150, 151,
750:     152, 153, 153, 154, 155, 155, 156, 156, 156, 156,
751:     156, 157, 158, 159, 159, 160, 160, 161, 161, 162,
752:     162, 162, 163, 164, 164, 165, 165, 166, 167, 168,
753:     169, 169, 170, 170, 170, 170, 170, 170, 170, 170,
754:     170, 170, 170, 170, 170, 170, 170, 170, 170, 170,
755:     170, 170, 171, 172, 173, 173, 174, 174, 174, 174,
756:     175, 175, 176, 176, 177, 178, 179, 179, 180, 180,
757:     181, 181, 182, 182, 183, 183, 184, 184, 184, 184,
758:     185, 186, 187, 187, 188, 188, 189, 189, 190, 190,
759:     190, 191, 191, 192, 192, 192
760: };
761:
762: /* YYR2[YYN] -- Number of symbols composing right hand side of rule YYN. */
763: static const unsigned char yyr2[] =
764: {
765:     0,  2,  8,  3,  0,  3,  0,  5,  3,  2,
766:     0,  3,  3,  3,  3,  3,  3,  1,  1,  3,
767:     6,  2,  0,  3,  3,  3,  3,  3,  3,  3,
768:     1,  3,  3,  3,  3,  3,  3,  7,  2,  0,
769:     1,  1,  1,  2,  0,  1,  3,  3,  3,  3,
770:     3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
771:     3,  3,  3,  3,  5,  2,  0,  3,  3,  3,
772:     3,  3,  5,  2,  0,  3,  3,  3,  2,  0,

```

```

773: 5, 3, 2, 0, 3, 3, 3, 2, 0, 6,
774: 1, 3, 3, 2, 0, 2, 0, 1, 1, 1,
775: 3, 3, 3, 3, 2, 0, 1, 1, 1, 7,
776: 1, 7, 3, 0, 2, 0, 3, 3, 3, 1,
777: 6, 2, 0, 7, 2, 0, 3, 3, 3, 3,
778: 3, 1, 6, 2, 0, 7, 3, 2, 0, 3,
779: 3, 3, 1, 2, 0, 1, 1, 5, 2, 2,
780: 2, 0, 3, 3, 3, 3, 3, 3, 3, 3,
781: 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
782: 3, 3, 2, 1, 2, 0, 1, 1, 1, 1,
783: 2, 0, 5, 7, 4, 3, 2, 0, 3, 3,
784: 2, 0, 3, 5, 2, 0, 3, 3, 3, 3,
785: 4, 3, 2, 0, 3, 3, 2, 0, 3, 5,
786: 3, 2, 0, 3, 3, 3
787: };
788:
789: /* YYDEFACT[STATE-NAME] -- Default rule to reduce with in state
790: STATE-NUM when YYTABLE doesn't specify something else to do. Zero
791: means the default is an error. */
792: static const unsigned char yydefact[] =
793: {
794: 0, 0, 0, 0, 1, 0, 4, 0, 6, 0,
795: 0, 0, 3, 0, 0, 5, 10, 2, 0, 0,
796: 22, 44, 88, 17, 18, 8, 0, 0, 0, 0,
797: 0, 0, 9, 0, 0, 0, 0, 0, 0, 0,
798: 0, 0, 79, 19, 0, 0, 0, 0, 0, 0,
799: 0, 0, 0, 0, 0, 0, 30, 21, 39, 36,
800: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
801: 0, 0, 0, 0, 0, 0, 0, 45, 43, 0,
802: 94, 87, 96, 11, 13, 12, 15, 14, 16, 0,
803: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
804: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
805: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
806: 7, 0, 144, 83, 0, 78, 23, 24, 25, 26,
807: 27, 28, 31, 29, 32, 33, 34, 35, 0, 66,
808: 74, 38, 41, 42, 40, 62, 46, 47, 48, 49,
809: 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
810: 60, 61, 91, 92, 0, 0, 0, 0, 93, 98,
811: 99, 97, 95, 105, 0, 20, 79, 0, 0, 0,
812: 0, 0, 0, 151, 0, 143, 145, 146, 0, 0,
813: 81, 0, 0, 0, 82, 37, 0, 63, 0, 0,

```

```

814:    0,  0, 65,  0, 71,  0,  0,  0, 73, 100,
815: 102, 103, 101, 149,  0,  0,  0,  0, 104, 108,
816: 107, 106, 79, 148,  0,  0,  0,  0,  0,  0,
817:    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
818:    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
819:    0,  0,  0,  0,  0,  0,  0, 150, 134, 122,
820: 134, 89, 173,  0, 175, 80, 84, 85, 86, 64,
821: 70, 67, 68, 69, 72, 77, 75, 76,  0,  0,
822:    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
823:    0,  0,  0,  0,  0,  0,  0,  0, 113, 113,
824: 122,  0, 172, 152, 153, 156, 155, 157, 162, 159,
825: 154, 164, 158, 165, 163, 161, 166, 160, 167, 168,
826: 169, 170, 171, 138, 115,  0, 110, 133, 125,  0,
827: 121, 113, 147, 187, 203, 178, 179, 174, 176, 191,
828: 177, 207,  0,  0,  0,  0,  0,  0,  0,  0,
829:    0,  0, 79, 136,  0,  0,  0, 137, 79, 112,
830:    0,  0,  0, 114, 132, 79,  0,  0,  0,  0,
831:    0, 124, 120,  0, 185,  0,  0, 186, 201,  0,
832:    0, 202,  0, 195, 190,  0, 212, 206, 142, 181,
833:    0,  0,  0, 119, 181,  0,  0,  0, 131, 181,
834:    0,  0,  0,  0,  0, 109,  0,  0,  0,  0,
835: 184,  0, 200,  0,  0, 139, 140, 141,  0, 116,
836: 117, 118,  0, 126, 127, 129, 128, 130, 188, 189,
837: 204, 205,  0, 192,  0,  0,  0,  0, 194, 210,
838: 208,  0,  0,  0, 211,  0,  0, 180,  0,  0,
839:    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
840: 135, 111, 123, 193, 196, 198, 197, 199, 209, 215,
841: 214, 213,  0,  0,  0, 182,  0, 183
842: };
843:
844: /* YYDEFGOTO[NTERM-NUM]. */
845: static const short int yydefgoto[] =
846: {
847:   -1,  2,  8, 11, 17, 18, 19, 32, 22, 23,
848:    33, 56, 57, 24, 102, 141, 34, 78, 142, 177,
849:   202, 143, 178, 208, 89, 125, 174, 194, 35, 81,
850:    82, 121, 122, 172, 168, 184, 218, 219, 325, 326,
851:   343, 363, 394, 220, 299, 330, 345, 371, 399, 221,
852:   298, 327, 342, 357, 389, 173, 185, 186, 187, 188,
853:   213, 257, 263, 264, 302, 337, 414, 447, 338, 339,
854:   348, 377, 350, 384, 411, 438, 340, 341, 349, 381,

```

```

855: 351, 387, 413, 444
856: };
857:
858: /* YYPACT[STATE-NUM] -- Index in YYTABLE of the portion describing
859: STATE-NUM. */
860: #define YYPACT_NINF -383
861: static const short int yypact[] =
862: {
863: 8, -6, 40, 42, -383, 4, -44, 58, -29, 28,
864: 76, -17, -383, 38, 110, -383, -383, -383, 13, 19,
865: -383, -383, -383, -383, -383, -383, 154, 155, 156, 157,
866: 158, 160, -383, 2, 97, 6, 118, 119, 121, 129,
867: 136, 143, -383, -383, 185, 193, 194, 198, 199, 200,
868: 201, 202, 203, 204, 205, 207, -383, -383, -383, -383,
869: 209, 213, 214, 216, 217, 218, 219, 220, 221, 222,
870: 223, 224, 225, 226, 227, 228, 229, -383, -383, 189,
871: -383, -383, 9, -383, -383, -383, -383, -383, -383, -30,
872: 184, 187, 188, 191, 192, 195, 196, 197, 206, 208,
873: 210, 211, 69, 212, 215, 230, 231, 232, 233, 234,
874: 235, 236, 237, 238, 239, 240, 241, 242, 243, 244,
875: -383, 16, -46, -383, 245, -383, -383, -383, -383, -383,
876: -383, -383, -383, -383, -383, -383, -383, -383, 246, -383,
877: -383, -383, -383, -383, -383, -383, -383, -383, -383, -383,
878: -383, -383, -383, -383, -383, -383, -383, -383, -383, -383,
879: -383, -383, -383, -383, 248, 249, 250, 251, -383, -383,
880: -383, -383, -383, 252, 139, -383, -383, 5, 0, 247,
881: 253, 254, 255, -383, 33, -383, -383, -383, 104, 159,
882: -383, 260, 262, 263, -383, 153, 180, -383, 264, 265,
883: 266, 267, -383, 268, -383, 269, 270, 271, -383, -383,
884: -383, -383, -383, 10, 272, 273, 274, 291, -383, -383,
885: -383, -383, -383, -383, 293, 256, 257, 258, 294, 259,
886: 261, 275, 276, 295, 277, 278, 279, 297, 298, 306,
887: 308, 309, 310, 311, 312, 313, 314, 315, 316, 317,
888: 318, 319, 320, 326, 327, 328, 329, -383, -383, -383,
889: -383, -383, 153, 322, -383, -383, -383, -383, -383, -383,
890: -383, -383, -383, -383, -383, -383, -383, -383, 285, 286,
891: 287, 288, 289, 290, 292, 296, 299, 300, 301, 302,
892: 303, 304, 305, 307, 321, 323, 324, 325, -13, -16,
893: 334, 336, 3, -383, -383, -383, -383, -383, -383, -383,
894: -383, -383, -383, -383, -383, -383, -383, -383, -383, -383,
895: -383, -383, -383, -383, -383, 161, -383, -383, -383, 164,

```

```

896:  -383, -16, -383, -383, -383, -383, -383, -383, -383, -383,
897:  -383, -383, 186, 190, 338, 152, 339, 167, 84, 151,
898:   134, 94, -383, -383, 341, 343, 344, -383, -383, -383,
899:   354, 355, 356, -383, -383, -383, 357, 358, 359, 360,
900:   361, -383, -383, 362, -383, 364, 365, -383, -383, 366,
901:   368, -383, 371, -383, -383, 372, -383, -383, 153, -383,
902:   330, 331, 332, 153, -383, 333, 335, 337, 153, -383,
903:   340, 342, 345, 346, 347, -383, 348, 349, 350, 351,
904:  -383, 141, -383, 78, 36, -383, -383, -383, -1, -383,
905:  -383, -383, 130, -383, -383, -383, -383, -383, -383, -383,
906:  -383, -383, 170, -383, 374, 375, 380, 382, -383, 146,
907:  -383, 384, 385, 387, -383, 369, 389, -383, 397, 398,
908:   399, 363, 367, 370, 373, 400, 376, 377, 378, 402,
909:  -383, -383, -383, -383, -383, -383, -383, -383, -383, -383,
910:  -383, -383, 379, 148, 174, -383, 403, -383
911: };
912:
913: /* YYPGOTO[NTERM-NUM]. */
914: static const short int yypgoto[] =
915: {
916:  -383, -383, -383, -383, -383, -383, -383, -383, -383, -383,
917:  -383, 381, -383, -383, -383, -383, -383, -383, -121, -383,
918:  -383, -119, -383, -383, -176, -74, -383, -383, -383, -383,
919:  -383, -383, -383, -383, -383, -383, -383, -383, -288, -383,
920:  -383, -383, -383, -383, -42, -383, -383, -383, -383, -383,
921:   -4, -383, -383, -383, -383, -383, -383, -383, -383, -383,
922:  -383, -383, -383, -383, -383, -383, -382, -383, -383, -383,
923:  -383, -383, -383, -383, -383, -383, -383, -383, -383, -383,
924:  -383, -383, -383, -383
925: };
926:
927: /* YYTABLE[YYPACT[STATE-NUM]]. What to do in state STATE-NUM. If
928:  positive, shift that token. If negative, reduce the rule which
929:  number is the opposite. If zero, do what YYDEFACHT says.
930:  If YYTABLE_NINF, syntax error. */
931: #define YYTABLE_NINF -91
932: static const short int yytable[] =
933: {
934:   195, 169, 328, 170, 203, 204, 42, 43, 323, 196,
935:   197, 329, 418, 79, 1, 333, -90, 422, 334, 445,
936:   162, 163, 20, 25, 3, 44, 45, 46, 144, 47,

```

937: 48, 49, 50, 237, 238, 239, 123, 240, 241, 164,
 938: 4, 165, 26, 347, 166, 5, 262, 27, 171, 28,
 939: 139, 140, 123, 6, 124, 7, 445, 324, 446, 205,
 940: 324, 9, 242, 243, 198, 244, 245, 246, 247, 248,
 941: 249, 10, 250, 251, 21, 80, 51, 12, -90, 13,
 942: 448, 14, 439, 440, 29, 30, 31, 15, 374, 52,
 943: 53, 54, 55, 167, 206, 207, 199, 200, 201, 139,
 944: 140, 58, 59, 214, 215, 216, 441, 375, 222, 223,
 945: 385, 386, 376, 252, 253, 254, 255, 256, 16, 217,
 946: 44, 45, 46, 442, 47, 48, 49, 60, 61, 62,
 947: 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
 948: 73, 74, 443, 189, 190, 432, 433, 382, 383, 449,
 949: 445, 123, 474, 475, 138, 378, 365, 36, 37, 38,
 950: 39, 40, 191, 41, 434, 139, 140, 83, 84, 435,
 951: 85, 51, 75, 76, 379, 366, 388, 367, 86, 380,
 952: 368, 335, 393, 336, 192, 87, 436, 437, 90, 398,
 953: 352, 353, 88, 120, 358, 359, 91, 92, 193, 369,
 954: 370, 93, 94, 95, 96, 97, 98, 99, 100, 354,
 955: 101, 355, 103, 360, 356, 361, 104, 105, 362, 106,
 956: 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
 957: 117, 118, 119, 126, 228, 123, 127, 128, 450, 344,
 958: 129, 130, 224, 346, 131, 132, 133, 373, 455, 175,
 959: 176, 179, 180, 181, 182, 134, 300, 135, 331, 136,
 960: 137, 145, 183, 225, 146, 226, 227, 229, 230, 231,
 961: 232, 0, 234, 235, 236, 476, 258, 259, 260, 147,
 962: 148, 149, 150, 151, 152, 153, 154, 155, 156, 157,
 963: 158, 159, 160, 161, 233, 261, 209, 265, 269, 274,
 964: 278, 279, 210, 211, 212, 266, 267, 268, 270, 280,
 965: 271, 281, 282, 283, 284, 285, 286, 287, 288, 289,
 966: 290, 291, 292, 293, 272, 273, 275, 276, 277, 294,
 967: 295, 296, 297, 301, 303, 304, 305, 306, 307, 308,
 968: 332, 309, 364, 372, 390, 310, 391, 392, 311, 312,
 969: 313, 314, 315, 316, 317, 323, 318, 395, 396, 397,
 970: 400, 401, 402, 403, 404, 0, 405, 406, 407, 408,
 971: 319, 409, 320, 321, 322, 410, 412, 451, 452, 415,
 972: 416, 417, 419, 453, 420, 454, 421, 456, 457, 423,
 973: 458, 424, 459, 460, 425, 426, 427, 428, 429, 430,
 974: 431, 461, 462, 463, 468, 472, 0, 477, 0, 0,
 975: 0, 0, 464, 0, 0, 77, 465, 0, 0, 466,
 976: 0, 0, 467, 0, 0, 469, 470, 471, 473
 977: };


```

978:
979: static const short int yycheck[] =
980: {
981: 176, 122, 18, 122, 4, 5, 4, 5, 21, 4,
982: 5, 299, 394, 7, 6, 12, 7, 399, 15, 20,
983: 4, 5, 9, 4, 30, 23, 24, 25, 102, 27,
984: 28, 29, 30, 23, 24, 25, 82, 27, 28, 23,
985: 0, 25, 23, 331, 28, 3, 222, 28, 122, 30,
986: 96, 97, 82, 49, 84, 99, 20, 73, 22, 59,
987: 73, 3, 52, 53, 59, 55, 56, 57, 58, 59,
988: 60, 100, 62, 63, 61, 69, 74, 49, 69, 3,
989: 81, 98, 4, 5, 65, 66, 67, 49, 4, 87,
990: 88, 89, 90, 77, 94, 95, 91, 92, 93, 96,
991: 97, 4, 5, 70, 71, 72, 28, 23, 4, 5,
992: 16, 17, 28, 103, 104, 105, 106, 107, 8, 86,
993: 23, 24, 25, 45, 27, 28, 29, 30, 31, 32,
994: 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
995: 43, 44, 64, 4, 5, 4, 5, 13, 14, 19,
996: 20, 82, 4, 5, 85, 4, 4, 3, 3, 3,
997: 3, 3, 23, 3, 23, 96, 97, 49, 49, 28,
998: 49, 74, 75, 76, 23, 23, 352, 25, 49, 28,
999: 28, 302, 358, 302, 45, 49, 45, 46, 3, 365,
1000: 4, 5, 49, 4, 4, 5, 3, 3, 59, 47,
1001: 48, 3, 3, 3, 3, 3, 3, 3, 3, 23,
1002: 3, 25, 3, 23, 28, 25, 3, 3, 28, 3,
1003: 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
1004: 3, 3, 3, 49, 54, 82, 49, 49, 68, 78,
1005: 49, 49, 83, 79, 49, 49, 49, 80, 102, 4,
1006: 4, 3, 3, 3, 3, 49, 260, 49, 300, 49,
1007: 49, 49, 10, 3, 49, 3, 3, 3, 3, 3,
1008: 3, -1, 3, 3, 3, 101, 4, 4, 4, 49,
1009: 49, 49, 49, 49, 49, 49, 49, 49, 49, 49,
1010: 49, 49, 49, 49, 26, 4, 49, 4, 4, 4,
1011: 3, 3, 49, 49, 49, 49, 49, 49, 49, 3,
1012: 49, 3, 3, 3, 3, 3, 3, 3, 3, 3,
1013: 3, 3, 3, 3, 49, 49, 49, 49, 49, 3,
1014: 3, 3, 3, 11, 49, 49, 49, 49, 49, 49,
1015: 4, 49, 4, 4, 3, 49, 3, 3, 49, 49,
1016: 49, 49, 49, 49, 49, 21, 49, 3, 3, 3,
1017: 3, 3, 3, 3, 3, -1, 4, 3, 3, 3,
1018: 49, 3, 49, 49, 49, 4, 4, 3, 3, 49,

```

```

1019: 49, 49, 49, 3, 49, 3, 49, 3, 3, 49,
1020: 3, 49, 23, 4, 49, 49, 49, 49, 49, 49,
1021: 49, 4, 4, 4, 4, 3, -1, 4, -1, -1,
1022: -1, -1, 49, -1, -1, 34, 49, -1, -1, 49,
1023: -1, -1, 49, -1, -1, 49, 49, 49, 49
1024: };
1025:
1026: /* YYSTOS[STATE-NUM] -- The (internal number of the) accessing
1027:  symbol of state STATE-NUM. */
1028: static const unsigned char yystos[] =
1029: {
1030: 0, 6, 110, 30, 0, 3, 49, 99, 111, 3,
1031: 100, 112, 49, 3, 98, 49, 8, 113, 114, 115,
1032: 9, 61, 117, 118, 122, 4, 23, 28, 30, 65,
1033: 66, 67, 116, 119, 125, 137, 3, 3, 3, 3,
1034: 3, 3, 4, 5, 23, 24, 25, 27, 28, 29,
1035: 30, 74, 87, 88, 89, 90, 120, 121, 4, 5,
1036: 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
1037: 40, 41, 42, 43, 44, 75, 76, 120, 126, 7,
1038: 69, 138, 139, 49, 49, 49, 49, 49, 49, 133,
1039: 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
1040: 3, 3, 123, 3, 3, 3, 3, 3, 3, 3,
1041: 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
1042: 4, 140, 141, 82, 84, 134, 49, 49, 49, 49,
1043: 49, 49, 49, 49, 49, 49, 49, 49, 85, 96,
1044: 97, 124, 127, 130, 134, 49, 49, 49, 49, 49,
1045: 49, 49, 49, 49, 49, 49, 49, 49, 49, 49,
1046: 49, 49, 4, 5, 23, 25, 28, 77, 143, 127,
1047: 130, 134, 142, 164, 135, 4, 4, 128, 131, 3,
1048: 3, 3, 3, 10, 144, 165, 166, 167, 168, 4,
1049: 5, 23, 45, 59, 136, 133, 4, 5, 59, 91,
1050: 92, 93, 129, 4, 5, 59, 94, 95, 132, 49,
1051: 49, 49, 49, 169, 70, 71, 72, 86, 145, 146,
1052: 152, 158, 4, 5, 83, 3, 3, 3, 54, 3,
1053: 3, 3, 3, 26, 3, 3, 3, 23, 24, 25,
1054: 27, 28, 52, 53, 55, 56, 57, 58, 59, 60,
1055: 62, 63, 103, 104, 105, 106, 107, 170, 4, 4,
1056: 4, 4, 133, 171, 172, 4, 49, 49, 49, 4,
1057: 49, 49, 49, 49, 4, 49, 49, 49, 3, 3,
1058: 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
1059: 3, 3, 3, 3, 3, 3, 3, 3, 159, 153,

```

```

1060: 159, 11, 173, 49, 49, 49, 49, 49, 49, 49,
1061: 49, 49, 49, 49, 49, 49, 49, 49, 49, 49,
1062: 49, 49, 49, 21, 73, 147, 148, 160, 18, 147,
1063: 154, 153, 4, 12, 15, 127, 130, 174, 177, 178,
1064: 185, 186, 161, 149, 78, 155, 79, 147, 179, 187,
1065: 181, 189, 4, 5, 23, 25, 28, 162, 4, 5,
1066: 23, 25, 28, 150, 4, 4, 23, 25, 28, 47,
1067: 48, 156, 4, 80, 4, 23, 28, 180, 4, 23,
1068: 28, 188, 13, 14, 182, 16, 17, 190, 133, 163,
1069: 3, 3, 3, 133, 151, 3, 3, 3, 133, 157,
1070: 3, 3, 3, 3, 3, 4, 3, 3, 3, 3,
1071: 4, 183, 4, 191, 175, 49, 49, 49, 175, 49,
1072: 49, 49, 175, 49, 49, 49, 49, 49, 49, 49,
1073: 49, 49, 4, 5, 23, 28, 45, 46, 184, 4,
1074: 5, 28, 45, 64, 192, 20, 22, 176, 81, 19,
1075: 68, 3, 3, 3, 3, 102, 3, 3, 3, 23,
1076: 4, 4, 4, 4, 49, 49, 49, 49, 4, 49,
1077: 49, 49, 3, 49, 4, 5, 101, 4
1078: };
1079:
1080: #if ! defined (YYSIZE_T) && defined (__SIZE_TYPE__)
1081: # define YYSIZE_T __SIZE_TYPE__
1082: #endif
1083: #if ! defined (YYSIZE_T) && defined (size_t)
1084: # define YYSIZE_T size_t
1085: #endif
1086: #if ! defined (YYSIZE_T)
1087: # if defined (__STDC__) || defined (__cplusplus)
1088: # include <stddef.h> /* INFRINGES ON USER NAME SPACE */
1089: # define YYSIZE_T size_t
1090: # endif
1091: #endif
1092: #if ! defined (YYSIZE_T)
1093: # define YYSIZE_T unsigned int
1094: #endif
1095:
1096: #define yyerrok      (yyerrstatus = 0)
1097: #define yyclearin (yychar = YYEMPTY)
1098: #define YYEMPTY      (-2)
1099: #define YYEOF        0
1100:

```

```

1101: #define YYACCEPT goto yyacceptlab
1102: #define YYABORT      goto yyabortlab
1103: #define YYERROR      goto yyerrorlab
1104:
1105:
1106: /* Like YYERROR except do call yyerror. This remains here temporarily
1107:  to ease the transition to the new meaning of YYERROR, for GCC.
1108:  Once GCC version 2 has supplanted version 1, this can go. */
1109:
1110: #define YYFAIL      goto yyerrlab
1111:
1112: #define YYRECOVERING() (!yyerrstatus)
1113:
1114: #define YYBACKUP(Token, Value)          \
1115: do                                     \
1116:   if (yychar == YYEMPTY && yylen == 1) \
1117:   {                                     \
1118:     yychar = (Token);                  \
1119:     yylval = (Value);                   \
1120:     yytoken = YYTRANSLATE (yychar);    \
1121:     YYPOPSTACK;                         \
1122:     goto yybackup;                      \
1123:   }                                     \
1124:   else                                 \
1125:   {                                     \
1126:     yyerror ("syntax error: cannot back up"); \
1127:     YYERROR;                             \
1128:   }                                     \
1129: while (0)
1130:
1131: #define YYTERROR 1
1132: #define YYERRCODE 256
1133:
1134: /* YYLLOC_DEFAULT -- Compute the default location (before the actions
1135:  are run). */
1136:
1137: #ifndef YYLLOC_DEFAULT
1138: # define YYLLOC_DEFAULT(Current, Rhs, N)      \
1139:   ((Current).first_line = (Rhs)[1].first_line, \
1140:    (Current).first_column = (Rhs)[1].first_column, \
1141:    (Current).last_line = (Rhs)[N].last_line, \

```

```

1142:   (Current).last_column = (Rhs)[N].last_column)
1143: #endif
1144:
1145: /* YYLEX -- calling `yylex' with the right arguments. */
1146:
1147: #ifdef YYLEX_PARAM
1148: # define YYLEX yylex (YYLEX_PARAM)
1149: #else
1150: # define YYLEX yylex ()
1151: #endif
1152:
1153: /* Enable debugging if requested. */
1154: #if YYDEBUG
1155:
1156: # ifndef YYFPRINTF
1157: #  include <stdio.h> /* INFRINGES ON USER NAME SPACE */
1158: #  define YYFPRINTF fprintf
1159: # endif
1160:
1161: # define YYDPRINTF(Args)      \
1162: do {                          \
1163:   if (yydebug)               \
1164:     YYFPRINTF Args;          \
1165: } while (0)
1166:
1167: # define YYDSYMPRINT(Args)    \
1168: do {                          \
1169:   if (yydebug)               \
1170:     yysymprint Args;          \
1171: } while (0)
1172:
1173: # define YYDSYMPRINTF(Title, Token, Value, Location)      \
1174: do {                                                        \
1175:   if (yydebug)                                             \
1176:     {                                                      \
1177:       YYFPRINTF (stderr, "%s ", Title);                   \
1178:       yysymprint (stderr,                                  \
1179:                   Token, Value); \
1180:       YYFPRINTF (stderr, " \n");                          \
1181:     }                                                      \
1182: } while (0)

```

```

1183:
1184: /*-----
1185: | yy_stack_print -- Print the state stack from its BOTTOM up to its |
1186: | TOP (included).                                         |
1187: `-----*/
1188:
1189: #if defined (__STDC__) || defined (__cplusplus)
1190: static void
1191: yy_stack_print (short int *bottom, short int *top)
1192: #else
1193: static void
1194: yy_stack_print (bottom, top)
1195:     short int *bottom;
1196:     short int *top;
1197: #endif
1198: {
1199:     YYFPRINTF (stderr, "Stack now");
1200:     for (/* Nothing. */; bottom <= top; ++bottom)
1201:         YYFPRINTF (stderr, " %d", *bottom);
1202:     YYFPRINTF (stderr, " \n");
1203: }
1204:
1205: # define YY_STACK_PRINT(Bottom, Top) \
1206: do { \
1207:     if (yydebug) \
1208:         yy_stack_print ((Bottom), (Top)); \
1209: } while (0)
1210:
1211:
1212: /*-----
1213: | Report that the YYRULE is going to be reduced. |
1214: `-----*/
1215:
1216: #if defined (__STDC__) || defined (__cplusplus)
1217: static void
1218: yy_reduce_print (int yyrule)
1219: #else
1220: static void
1221: yy_reduce_print (yyrule)
1222:     int yyrule;
1223: #endif

```

```

1224: {
1225:   int yyi;
1226:   unsigned int yyno = yyrule[yyrline];
1227:   YYFPRINTF (stderr, "Reducing stack by rule %d (line %u), ",
1228:             yyrule - 1, yyno);
1229:   /* Print the symbols being reduced, and their result. */
1230:   for (yyi = yyprhs[yyrule]; 0 <= yyrhs[yyi]; yyi++)
1231:     YYFPRINTF (stderr, "%s ", yytname [yyrhs[yyi]]);
1232:   YYFPRINTF (stderr, "-> %s \n", yytname [yyr1[yyrule]]);
1233: }
1234:
1235: # define YY_REDUCE_PRINT(Rule)      \
1236: do {                                \
1237:   if (yydebug)                      \
1238:     yy_reduce_print (Rule);          \
1239: } while (0)
1240:
1241: /* Nonzero means print parse trace.  It is left uninitialized so that
1242:    multiple parsers can coexist.  */
1243: int yydebug;
1244: #else /* !YYDEBUG */
1245: # define YYDPRINTF(Args)
1246: # define YYDSYMPRINT(Args)
1247: # define YYDSYMPRINTF(Title, Token, Value, Location)
1248: # define YY_STACK_PRINT(Bottom, Top)
1249: # define YY_REDUCE_PRINT(Rule)
1250: #endif /* !YYDEBUG */
1251:
1252:
1253: /* YYINITDEPTH -- initial size of the parser's stacks.  */
1254: #ifndef YYINITDEPTH
1255: # define YYINITDEPTH 200
1256: #endif
1257:
1258: /* YYMAXDEPTH -- maximum size the stacks can grow to (effective only
1259:    if the built-in stack extension method is used).
1260:
1261:    Do not make this value too large; the results are undefined if
1262:    SIZE_MAX < YYSTACK_BYTES (YYMAXDEPTH)
1263:    evaluated with infinite-precision integer arithmetic.  */
1264:

```

```

1265: #if defined (YYMAXDEPTH) && YYMAXDEPTH == 0
1266: # undef YYMAXDEPTH
1267: #endif
1268:
1269: #ifndef YYMAXDEPTH
1270: # define YYMAXDEPTH 10000
1271: #endif
1272:
1275: #if YYERROR_VERBOSE
1276:
1277: # ifndef yystrlen
1278: #  if defined (__GLIBC__) && defined (_STRING_H)
1279: #   define yystrlen strlen
1280: #  else
1281: /* Return the length of YYSTR. */
1282: static YYSIZE_T
1283: #   if defined (__STDC__) || defined (__cplusplus)
1284: yystrlen (const char *yystr)
1285: #   else
1286: yystrlen (yystr)
1287:     const char *yystr;
1288: #   endif
1289: {
1290:   register const char *yys = yystr;
1291:
1292:   while (*yys++ != '\0')
1293:     continue;
1294:
1295:   return YYS - yystr - 1;
1296: }
1297: # endif
1298: # endif
1299:
1300: # ifndef yystpcpy
1301: #  if defined (__GLIBC__) && defined (_STRING_H) && defined (_GNU_SOURCE)
1302: #   define yystpcpy stpcpy
1303: #  else
1304: /* Copy YYSRC to YYDEST, returning the address of the terminating '\0' in
1305:   YYDEST. */
1306: static char *
1307: #   if defined (__STDC__) || defined (__cplusplus)

```



```

1308: yystpcpy (char *yydest, const char *yysrc)
1309: # else
1310: yystpcpy (yydest, yysrc)
1311:     char *yydest;
1312:     const char *yysrc;
1313: # endif
1314: {
1315:     register char *yyd = yydest;
1316:     register const char *yys = yysrc;
1317:
1318:     while ((*yyd++ = *yys++) != '\0')
1319:         continue;
1320:
1321:     return yyd - 1;
1322: }
1323: # endif
1324: # endif
1325:
1326: #endif /* !YYERROR_VERBOSE */
1329:
1330: #if YYDEBUG
1331: /*-----
1332: | Print this symbol on YYOUTPUT. |
1333: `-----*/
1334:
1335: #if defined (__STDC__) || defined (__cplusplus)
1336: static void
1337: yysymprint (FILE *yyoutput, int yytype, YYSTYPE *yyvaluep)
1338: #else
1339: static void
1340: yysymprint (yyoutput, yytype, yyvaluep)
1341:     FILE *yyoutput;
1342:     int yytype;
1343:     YYSTYPE *yyvaluep;
1344: #endif
1345: {
1346:     /* Pacify "unused variable" warnings. */
1347:     (void) yyvaluep;
1348:
1349:     if (yytype < YYNTOKENS)
1350:     {

```

```

1351:   YYFPRINTF (yyoutput, "token %s (", yyname[yytype]);
1352: # ifdef YYPRINT
1353:   YYPRINT (yyoutput, yytoknum[yytype], *yyvaluep);
1354: # endif
1355: }
1356: else
1357:   YYFPRINTF (yyoutput, "nterm %s (", yyname[yytype]);
1358:
1359: switch (yytype)
1360: {
1361:   default:
1362:     break;
1363: }
1364: YYFPRINTF (yyoutput, ")");
1365: }
1366:
1367: #endif /* ! YYDEBUG */
1368: /*-----
1369: | Release the memory associated to this symbol. |
1370: `-----*/
1371:
1372: #if defined (__STDC__) || defined (__cplusplus)
1373: static void
1374: yydestruct (int yytype, YYSTYPE *yyvaluep)
1375: #else
1376: static void
1377: yydestruct (yytype, yyvaluep)
1378:   int yytype;
1379:   YYSTYPE *yyvaluep;
1380: #endif
1381: {
1382:   /* Pacify ``unused variable" warnings. */
1383:   (void) yyvaluep;
1384:
1385: switch (yytype)
1386: {
1387:
1388:   default:
1389:     break;
1390: }
1391: }

```

```

1392:
1393:
1394: /* Prevent warnings from -Wmissing-prototypes. */
1395:
1396: #ifdef YYPARSE_PARAM
1397: # if defined (__STDC__) || defined (__cplusplus)
1398: int yyparse (void *YYPARSE_PARAM);
1399: # else
1400: int yyparse ();
1401: # endif
1402: #else /* ! YYPARSE_PARAM */
1403: # if defined (__STDC__) || defined (__cplusplus)
1404: int yyparse (void);
1405: # else
1406: int yyparse ();
1407: # endif
1408: #endif /* ! YYPARSE_PARAM */
1409:
1410:
1411:
1412: /* The lookahead symbol. */
1413: int yychar;
1414:
1415: /* The semantic value of the lookahead symbol. */
1416: YYSTYPE yylval;
1417:
1418: /* Number of syntax errors so far. */
1419: int yynerrs;
1420:
1421:
1422:
1423: /*-----
1424: | yyparse. |
1425: `-----*/
1426:
1427: #ifdef YYPARSE_PARAM
1428: # if defined (__STDC__) || defined (__cplusplus)
1429: int yyparse (void *YYPARSE_PARAM)
1430: # else
1431: int yyparse (YYPARSE_PARAM)
1432: void *YYPARSE_PARAM;

```

```

1433: # endif
1434: #else /* ! YYPARSE_PARAM */
1435: #if defined (__STDC__) || defined (__cplusplus)
1436: int
1437: yyparse (void)
1438: #else
1439: int
1440: yyparse ()
1441:
1442: #endif
1443: #endif
1444: {
1445:
1446:  register int yystate;
1447:  register int yyn;
1448:  int yyresult;
1449:  /* Number of tokens to shift before error messages enabled. */
1450:  int yyerrstatus;
1451:  /* Lookahead token as an internal (translated) token number. */
1452:  int yytoken = 0;
1453:
1454:  /* Three stacks and their tools:
1455:   `yyss': related to states,
1456:   `yyvs': related to semantic values,
1457:   `yyls': related to locations.
1458:
1459:   Refer to the stacks thru separate pointers, to allow yyoverflow
1460:   to reallocate them elsewhere.  */
1461:
1462:  /* The state stack.  */
1463:  short int yyssa[YYINITDEPTH];
1464:  short int *yyss = yyssa;
1465:  register short int *yyssp;
1466:
1467:  /* The semantic value stack.  */
1468:  YYSTYPE yyvsa[YYINITDEPTH];
1469:  YYSTYPE *yyvs = yyvsa;
1470:  register YYSTYPE *yyvsp;
1471:
1472:
1473:

```

```

1474: #define YYPOPSTACK (yyvsp--, yyssp--)
1475:
1476: YYSIZE_T yystacksize = YYINITDEPTH;
1477:
1478: /* The variables used to return semantic value and location from the
1479:    action routines. */
1480: YYSTYPE yyval;
1481:
1482:
1483: /* When reducing, the number of symbols on the RHS of the reduced
1484:    rule. */
1485: int yylen;
1486:
1487: YYDPRINTF ((stderr, "Starting parse \n"));
1488:
1489: yystate = 0;
1490: yyerrstatus = 0;
1491: yynerrs = 0;
1492: yychar = YYEMPTY;      /* Cause a token to be read. */
1493:
1494: /* Initialize stack pointers.
1495:    Waste one element of value and location stack
1496:    so that they stay on the same level as the state stack.
1497:    The wasted elements are never initialized. */
1498:
1499: yyssp = yyss;
1500: yyvsp = yyvs;
1501:
1502:
1503: goto yysetstate;
1504:
1505: /*-----
1506: | yynewstate -- Push a new state, which is found in yystate. |
1507: `-----*/
1508: yynewstate:
1509: /* In all cases, when you get here, the value and location stacks
1510:    have just been pushed. so pushing a state here evens the stacks.
1511:    */
1512: yyssp++;
1513:
1514: yysetstate:

```

```

1515: *yyssp = yystate;
1516:
1517: if (yyss + yystacksize - 1 <= yyssp)
1518: {
1519:     /* Get the current used size of the three stacks, in elements. */
1520:     YYSIZE_T yysize = yyssp - yyss + 1;
1521:
1522: #ifdef yyoverflow
1523:     {
1524:         /* Give user a chance to reallocate the stack. Use copies of
1525:            these so that the &'s don't force the real ones into
1526:            memory. */
1527:         YYSTYPE *yyvs1 = yyvs;
1528:         short int *yyss1 = yyss;
1529:
1530:
1531:         /* Each stack pointer address is followed by the size of the
1532:            data in use in that stack, in bytes. This used to be a
1533:            conditional around just the two extra args, but that might
1534:            be undefined if yyoverflow is a macro. */
1535:         yyoverflow ("parser stack overflow",
1536:                     &yyss1, yysize * sizeof (*yyssp),
1537:                     &yyvs1, yysize * sizeof (*yyvsp),
1538:
1539:                     &yystacksize);
1540:
1541:         yyss = yyss1;
1542:         yyvs = yyvs1;
1543:     }
1544: #else /* no yyoverflow */
1545: # ifdef YYSTACK_RELOCATE
1546:     goto yyoverflowlab;
1547: # else
1548:     /* Extend the stack our own way. */
1549:     if (YYMAXDEPTH <= yystacksize)
1550:         goto yyoverflowlab;
1551:     yystacksize *= 2;
1552:     if (YYMAXDEPTH < yystacksize)
1553:         yystacksize = YYMAXDEPTH;
1554:
1555:     {

```

```

1556: short int *yyss1 = yyss;
1557: union yyalloc *yyptr =
1558:     (union yyalloc *) YYSTACK_ALLOC (YYSTACK_BYTES (yystacksize));
1559: if (! yyptr)
1560:     goto yyoverflowlab;
1561: YYSTACK_RELOCATE (yyss);
1562: YYSTACK_RELOCATE (yyvs);
1563:
1564: # undef YYSTACK_RELOCATE
1565: if (yyss1 != yyssa)
1566:     YYSTACK_FREE (yyss1);
1567: }
1568: # endif
1569: #endif /* no yyoverflow */
1570:
1571: yyssp = yyss + yysize - 1;
1572: yyvsp = yyvs + yysize - 1;
1573:
1574:
1575: YYDPRINTF ((stderr, "Stack size increased to %lu \n",
1576:     (unsigned long int) yystacksize));
1577:
1578: if (yyss + yystacksize - 1 <= yyssp)
1579: YYABORT;
1580: }
1581:
1582: YYDPRINTF ((stderr, "Entering state %d \n", yystate));
1583:
1584: goto yybackup;
1585:
1586: /*-----,
1587: |yybackup. |
1588: `-----*/
1589: yybackup;
1590:
1591: /* Do appropriate processing given the current state. */
1592: /* Read a lookahead token if we need one and don't already have one. */
1593: /* yyresume: */
1594:
1595: /* First try to decide what to do without reference to lookahead token. */
1596:

```

```

1597: yyn = yypact[yystate];
1598: if (yyn == YYPACT_NINF)
1599:     goto yydefault;
1600:
1601: /* Not known => get a lookahead token if don't already have one. */
1602:
1603: /* YYCHAR is either YYEMPTY or YYEOF or a valid lookahead symbol. */
1604: if (yychar == YYEMPTY)
1605:     {
1606:         YYDPRINTF ((stderr, "Reading a token: "));
1607:         yychar = YYLEX;
1608:     }
1609:
1610: if (yychar <= YYEOF)
1611:     {
1612:         yychar = yytoken = YYEOF;
1613:         YYDPRINTF ((stderr, "Now at end of input. \n"));
1614:     }
1615: else
1616:     {
1617:         yytoken = YYTRANSLATE (yychar);
1618:         YYDSYMPRINTF ("Next token is", yytoken, &yylval, &yyloc);
1619:     }
1620:
1621: /* If the proper action on seeing token YYTOKEN is to reduce or to
1622:    detect an error, take that action. */
1623: yyn += yytoken;
1624: if (yyn < 0 || YYLAST < yyn || yycheck[yyn] != yytoken)
1625:     goto yydefault;
1626: yyn = yytable[yyn];
1627: if (yyn <= 0)
1628:     {
1629:         if (yyn == 0 || yyn == YYTABLE_NINF)
1630:             goto yyerrlab;
1631:         yyn = -yyn;
1632:         goto yyreduce;
1633:     }
1634:
1635: if (yyn == YYFINAL)
1636:     YYACCEPT;
1637:

```



```

1638: /* Shift the lookahead token. */
1639: YYDPRINTF ((stderr, "Shifting token %s, ", yytnamex[yytoken]));
1640:
1641: /* Discard the token being shifted unless it is eof. */
1642: if (yychar != YYEOF)
1643:   yychar = YYEMPTY;
1644:
1645: *++yyvsp = yylval;
1646:
1647:
1648: /* Count tokens shifted since error; after three, turn off error
1649:   status. */
1650: if (yyerrstatus)
1651:   yyerrstatus--;
1652:
1653: yystate = yyn;
1654: goto yynewstate;
1655:
1656:
1657: /*-----
1658: | yydefault -- do the default action for the current state. |
1659: `-----*/
1660: yydefault:
1661:   yyn = yydefact[yystate];
1662:   if (yyn == 0)
1663:     goto yyerrlab;
1664:   goto yyreduce;
1665:
1666:
1667: /*-----
1668: | yyreduce -- Do a reduction. |
1669: `-----*/
1670: yyreduce:
1671:   /* yyn is the number of a rule to reduce with. */
1672:   yrlen = yyr2[yyn];
1673:
1674:   /* If YYLEN is nonzero, implement the default value of the action:
1675:    `$$ = $1'.
1676:
1677:    Otherwise, the following line sets YYVAL to garbage.
1678:    This behavior is undocumented and Bison

```

```

1679:  users should not rely upon it. Assigning to YYVAL
1680:  unconditionally makes the parser a bit smaller, and it avoids a
1681:  GCC warning that YYVAL may be used uninitialized.  */
1682:  yyval = yyvsp[1-yylen];
1683:
1684:
1685:  YY_REDUCE_PRINT (yyn);
1686:  switch (yyn)
1687:  {
1688:      case 2:
1689: #line 93 "xTEDS.y"
1690:  {
1691:          int compare = 1;
1692:          if(strcmp("1.0",yyvsp[-4].str)!=0)
1693:              compare = 0;
1694:          free(yyvsp[-4].str);
1695:          compare = yyvsp[-3].integer;
1696:          if(compare == 1)
1697:              result = yyvsp[0].xteds;
1698:          else
1699:              result = NULL;
1700:      };}
1701:  break;
1702:
1703:  case 3:
1704: #line 107 "xTEDS.y"
1705:  {
1706:          int compare = 1;
1707:
1708:          if(strcmp("utf-8",yyvsp[0].str)!=0 && strcmp("UTF-8",yyvsp[0].str)!=0)
1709:              compare = 0;
1710:          free(yyvsp[0].str);
1711:          yyval.integer = compare;
1712:      };}
1713:  break;
1714:
1715:  case 4:
1716: #line 116 "xTEDS.y"
1717:  {
1718:          yyval.integer = 1;
1719:      };}

```

```

1720:  break;
1721:
1722:  case 5:
1723: #line 122 "xTEDS.y"
1724:  {
1725:          free(yyvsp[0].str);
1726:      ;}
1727:  break;
1728:
1729:  case 7:
1730: #line 134 "xTEDS.y"
1731:  {
1732:          yyval.xteds = add_references(yyvsp[-4].xteds,yyvsp[-3].attr,yyvsp[-2].interface);
1733:      ;}
1734:  break;
1735:
1736:  case 8:
1737: #line 140 "xTEDS.y"
1738:  {
1739:          yyval.xteds=yyvsp[-1].xteds;
1740:      ;}
1741:  break;
1742:
1743:  case 9:
1744: #line 146 "xTEDS.y"
1745:  {
1746:          yyval.xteds = merge_xteds(yyvsp[-1].xteds,yyvsp[0].xteds);
1747:      ;}
1748:  break;
1749:
1750:  case 10:
1751: #line 150 "xTEDS.y"
1752:  {
1753:          xteds* temp;
1754:          temp = new_xteds();
1755:          yyval.xteds=temp;
1756:      ;}
1757:  break;
1758:
1759:  case 11:
1760: #line 158 "xTEDS.y"

```

```

1761:  {
1762:      xteds* temp;
1763:      temp = new_xteds();
1764:      temp->name = yyvsp[0].str;
1765:      yyval.xteds = temp;
1766:      ;}
1767:  break;
1768:
1769:  case 12:
1770: #line 165 "xTEDS.y"
1771:  {
1772:      xteds* temp;
1773:      temp = new_xteds();
1774:      temp->version = yyvsp[0].str;
1775:      yyval.xteds = temp;
1776:      ;}
1777:  break;
1778:
1779:  case 13:
1780: #line 172 "xTEDS.y"
1781:  {
1782:      xteds* temp;
1783:      temp = new_xteds();
1784:      temp->description = yyvsp[0].str;
1785:      yyval.xteds = temp;
1786:      ;}
1787:  break;
1788:
1789:  case 14:
1790: #line 179 "xTEDS.y"
1791:  {
1792:      xteds* temp;
1793:      temp = new_xteds();
1794:      temp->xmlns = yyvsp[0].str;
1795:      yyval.xteds = temp;
1796:      ;}
1797:  break;
1798:
1799:  case 15:
1800: #line 186 "xTEDS.y"
1801:  {

```

```

1802:         xteds* temp;
1803:         temp = new_xteds();
1804:         temp->schema_location = yyvsp[0].str;
1805:         yyval.xteds = temp;
1806:     ;}
1807:     break;
1808:
1809: case 16:
1810: #line 193 "xTEDS.y"
1811:     {
1812:         xteds* temp;
1813:         temp = new_xteds();
1814:         temp->xmlns_xsi = yyvsp[0].str;
1815:         yyval.xteds = temp;
1816:     ;}
1817:     break;
1818:
1819: case 17:
1820: #line 207 "xTEDS.y"
1821:     {
1822:         yyval.attr = yyvsp[0].attr;
1823:     ;}
1824:     break;
1825:
1826: case 18:
1827: #line 211 "xTEDS.y"
1828:     {
1829:         yyval.attr = yyvsp[0].attr;
1830:     ;}
1831:     break;
1832:
1833: case 19:
1834: #line 217 "xTEDS.y"
1835:     {
1836:         yyval.attr = yyvsp[-1].attr;
1837:     ;}
1838:     break;
1839:
1840: case 20:
1841: #line 221 "xTEDS.y"
1842:     {

```

```

1843:         app_dev_attr* temp;
1844:         temp = new_app_dev_attr();
1845:         temp->qualifiers = yyvsp[-2].qual;
1846:         yyval.attr = merge_app_dev_attr(yyvsp[-4].attr,temp);
1847:     ;}
1848:     break;
1849:
1850: case 21:
1851: #line 230 "xTEDS.y"
1852:     {
1853:         yyval.attr = merge_app_dev_attr(yyvsp[-1].attr,yyvsp[0].attr);
1854:     ;}
1855:     break;
1856:
1857: case 22:
1858: #line 234 "xTEDS.y"
1859:     {
1860:         yyval.attr = NULL;
1861:     ;}
1862:     break;
1863:
1864: case 23:
1865: #line 240 "xTEDS.y"
1866:     {
1867:         app_dev_attr* temp;
1868:         temp = new_app_dev_attr();
1869:         temp->name = yyvsp[0].str;
1870:         yyval.attr = temp;
1871:     ;}
1872:     break;
1873:
1874: case 24:
1875: #line 247 "xTEDS.y"
1876:     {
1877:         app_dev_attr* temp;
1878:         temp = new_app_dev_attr();
1879:         temp->kind = yyvsp[0].str;
1880:         yyval.attr = temp;
1881:     ;}
1882:     break;
1883:

```

```

1884: case 25:
1885: #line 254 "xTEDS.y"
1886: {
1887:     app_dev_attr* temp;
1888:     temp = new_app_dev_attr();
1889:     temp->id = yyvsp[0].str;
1890:     yyval.attr = temp;
1891:     ;}
1892: break;
1893:
1894: case 26:
1895: #line 261 "xTEDS.y"
1896: {
1897:     app_dev_attr* temp;
1898:     temp = new_app_dev_attr();
1899:     temp->qualifier = yyvsp[0].str;
1900:     printf("Qualifier field has been deprecated! \n");
1901:     yyval.attr = temp;
1902:     ;}
1903: break;
1904:
1905: case 27:
1906: #line 269 "xTEDS.y"
1907: {
1908:     app_dev_attr* temp;
1909:     temp = new_app_dev_attr();
1910:     temp->description = yyvsp[0].str;
1911:     yyval.attr = temp;
1912:     ;}
1913: break;
1914:
1915: case 28:
1916: #line 276 "xTEDS.y"
1917: {
1918:     app_dev_attr* temp;
1919:     temp = new_app_dev_attr();
1920:     temp->manufacturer_id = yyvsp[0].str;
1921:     yyval.attr = temp;
1922:     ;}
1923: break;
1924:

```

```

1925: case 29:
1926: #line 283 "xTEDS.y"
1927: {
1928:     app_dev_attr* temp;
1929:     temp = new_app_dev_attr();
1930:     temp->componentKey = yyvsp[0].str;
1931:     yyval.attr = temp;
1932:     ;}
1933: break;
1934:
1935: case 30:
1936: #line 292 "xTEDS.y"
1937: {
1938:     yyval.attr = yyvsp[0].attr;
1939:     ;}
1940: break;
1941:
1942: case 31:
1943: #line 296 "xTEDS.y"
1944: {
1945:     app_dev_attr* temp;
1946:     temp = new_app_dev_attr();
1947:     temp->version = yyvsp[0].str;
1948:     yyval.attr = temp;
1949:     ;}
1950: break;
1951:
1952: case 32:
1953: #line 303 "xTEDS.y"
1954: {
1955:     app_dev_attr* temp;
1956:     temp = new_app_dev_attr();
1957:     temp->memory_minimum = yyvsp[0].str;
1958:     yyval.attr = temp;
1959:     ;}
1960: break;
1961:
1962: case 33:
1963: #line 310 "xTEDS.y"
1964: {
1965:     app_dev_attr* temp;

```



```

1966:         temp = new_app_dev_attr();
1967:         temp->operating_system = yyvsp[0].str;
1968:         yyval.attr = temp;
1969:     ;}
1970:     break;
1971:
1972: case 34:
1973: #line 317 "xTEDS.y"
1974:     {
1975:         app_dev_attr* temp;
1976:         temp = new_app_dev_attr();
1977:         temp->path_for_assembly = yyvsp[0].str;
1978:         yyval.attr = temp;
1979:     ;}
1980:     break;
1981:
1982: case 35:
1983: #line 324 "xTEDS.y"
1984:     {
1985:         app_dev_attr* temp;
1986:         temp = new_app_dev_attr();
1987:         temp->path_on_spacecraft = yyvsp[0].str;
1988:         yyval.attr = temp;
1989:     ;}
1990:     break;
1991:
1992: case 36:
1993: #line 338 "xTEDS.y"
1994:     {
1995:         yyval.attr = yyvsp[-1].attr;
1996:     ;}
1997:     break;
1998:
1999: case 37:
2000: #line 342 "xTEDS.y"
2001:     {
2002:         app_dev_attr* temp;
2003:         temp = new_app_dev_attr();
2004:         temp->qualifiers = yyvsp[0].qual;
2005:         merge_app_dev_attr(yyvsp[-5].attr,temp);
2006:         yyval.attr = merge_app_dev_attr(yyvsp[-5].attr,yyvsp[-3].attr);

```

```

2007:         ;}
2008:     break;
2009:
2010: case 38:
2011: #line 352 "xTEDS.y"
2012:     {
2013:         yyval.attr = merge_app_dev_attr(yyvsp[-1].attr,yyvsp[0].attr);
2014:     ;}
2015:     break;
2016:
2017: case 39:
2018: #line 356 "xTEDS.y"
2019:     {
2020:         yyval.attr = NULL;
2021:     ;}
2022:     break;
2023:
2024: case 40:
2025: #line 362 "xTEDS.y"
2026:     {
2027:         app_dev_attr* temp = new_app_dev_attr();
2028:         temp->qualifiers = yyvsp[0].qual;
2029:         yyval.attr = temp;
2030:     ;}
2031:     break;
2032:
2033: case 41:
2034: #line 368 "xTEDS.y"
2035:     {
2036:         delete_location(yyvsp[0].location); /*Location not used*/
2037:         yyval.attr = NULL;
2038:     ;}
2039:     break;
2040:
2041: case 42:
2042: #line 373 "xTEDS.y"
2043:     {
2044:         delete_orientation(yyvsp[0].orientation);/*Orientation not used*/
2045:         yyval.attr = NULL;
2046:     ;}
2047:     break;

```

```

2048:
2049: case 43:
2050: #line 380 "xTEDS.y"
2051: {
2052:     yyval.attr = merge_app_dev_attr(yyvsp[-1].attr,yyvsp[0].attr);
2053:     ;}
2054: break;
2055:
2056: case 44:
2057: #line 384 "xTEDS.y"
2058: {
2059:     yyval.attr = NULL;
2060:     ;}
2061: break;
2062:
2063: case 45:
2064: #line 390 "xTEDS.y"
2065: {
2066:     yyval.attr = yyvsp[0].attr;
2067:     ;}
2068: break;
2069:
2070: case 46:
2071: #line 394 "xTEDS.y"
2072: {
2073:     app_dev_attr* temp;
2074:     temp = new_app_dev_attr();
2075:     temp->model_id = yyvsp[0].str;
2076:     yyval.attr = temp;
2077:     ;}
2078: break;
2079:
2080: case 47:
2081: #line 401 "xTEDS.y"
2082: {
2083:     app_dev_attr* temp;
2084:     temp = new_app_dev_attr();
2085:     temp->version_letter = yyvsp[0].str;
2086:     yyval.attr = temp;
2087:     ;}
2088: break;

```

```

2089:
2090: case 48:
2091: #line 408 "xTEDS.y"
2092: {
2093:     app_dev_attr* temp;
2094:     temp = new_app_dev_attr();
2095:     temp->serial_number = yyvsp[0].str;
2096:     yyval.attr = temp;
2097:     ;}
2098: break;
2099:
2100: case 49:
2101: #line 415 "xTEDS.y"
2102: {
2103:     app_dev_attr* temp;
2104:     temp = new_app_dev_attr();
2105:     temp->calibration_date = yyvsp[0].str;
2106:     yyval.attr = temp;
2107:     ;}
2108: break;
2109:
2110: case 50:
2111: #line 422 "xTEDS.y"
2112: {
2113:     app_dev_attr* temp;
2114:     temp = new_app_dev_attr();
2115:     temp->sensitivity_at_reference = yyvsp[0].str;
2116:     yyval.attr = temp;
2117:     ;}
2118: break;
2119:
2120: case 51:
2121: #line 429 "xTEDS.y"
2122: {
2123:     app_dev_attr* temp;
2124:     temp = new_app_dev_attr();
2125:     temp->reference_frequency = yyvsp[0].str;
2126:     yyval.attr = temp;
2127:     ;}
2128: break;
2129:

```

```

2130: case 52:
2131: #line 436 "xTEDS.y"
2132: {
2133:     app_dev_attr* temp;
2134:     temp = new_app_dev_attr();
2135:     temp->reference_temperature = yyvsp[0].str;
2136:     yyval.attr = temp;
2137:     ;}
2138: break;
2139:
2140: case 53:
2141: #line 443 "xTEDS.y"
2142: {
2143:     app_dev_attr* temp;
2144:     temp = new_app_dev_attr();
2145:     temp->measurement_range = yyvsp[0].str;
2146:     yyval.attr = temp;
2147:     ;}
2148: break;
2149:
2150: case 54:
2151: #line 450 "xTEDS.y"
2152: {
2153:     app_dev_attr* temp;
2154:     temp = new_app_dev_attr();
2155:     temp->electrical_output = yyvsp[0].str;
2156:     yyval.attr = temp;
2157:     ;}
2158: break;
2159:
2160: case 55:
2161: #line 457 "xTEDS.y"
2162: {
2163:     app_dev_attr* temp;
2164:     temp = new_app_dev_attr();
2165:     temp->quality_factor = yyvsp[0].str;
2166:     yyval.attr = temp;
2167:     ;}
2168: break;
2169:
2170: case 56:

```

```

2171: #line 464 "xTEDS.y"
2172:  {
2173:      app_dev_attr* temp;
2174:      temp = new_app_dev_attr();
2175:      temp->temperature_coefficient = yyvsp[0].str;
2176:      yyval.attr = temp;
2177:      ;}
2178:  break;
2179:
2180: case 57:
2181: #line 471 "xTEDS.y"
2182:  {
2183:      app_dev_attr* temp;
2184:      temp = new_app_dev_attr();
2185:      temp->direction_xyz = yyvsp[0].str;
2186:      yyval.attr = temp;
2187:      ;}
2188:  break;
2189:
2190: case 58:
2191: #line 478 "xTEDS.y"
2192:  {
2193:      app_dev_attr* temp;
2194:      temp = new_app_dev_attr();
2195:      temp->cal_due_date = yyvsp[0].str;
2196:      yyval.attr = temp;
2197:      ;}
2198:  break;
2199:
2200: case 59:
2201: #line 485 "xTEDS.y"
2202:  {
2203:      app_dev_attr* temp;
2204:      temp = new_app_dev_attr();
2205:      temp->power_requirements = yyvsp[0].str;
2206:      yyval.attr = temp;
2207:      ;}
2208:  break;
2209:
2210: case 60:
2211: #line 492 "xTEDS.y"

```

```

2212:  {
2213:      app_dev_attr* temp;
2214:      temp = new_app_dev_attr();
2215:      temp->spa_u_hub = yyvsp[0].str;
2216:      yyval.attr = temp;
2217:  };}
2218:  break;
2219:
2220: case 61:
2221: #line 499 "xTEDS.y"
2222:  {
2223:      app_dev_attr* temp;
2224:      temp = new_app_dev_attr();
2225:      temp->spa_u_port = yyvsp[0].str;
2226:      yyval.attr = temp;
2227:  };}
2228:  break;
2229:
2230: case 62:
2231: #line 506 "xTEDS.y"
2232:  {
2233:      app_dev_attr* temp;
2234:      temp = new_app_dev_attr();
2235:      temp->version = yyvsp[0].str;
2236:      yyval.attr = temp;
2237:  };}
2238:  break;
2239:
2240: case 63:
2241: #line 520 "xTEDS.y"
2242:  {
2243:      yyval.location = yyvsp[-1].location;
2244:  };}
2245:  break;
2246:
2247: case 64:
2248: #line 524 "xTEDS.y"
2249:  {
2250:      yyval.location = yyvsp[-3].location;
2251:  };}
2252:  break;

```

```

2253:
2254: case 65:
2255: #line 530 "xTEDS.y"
2256: {
2257:     yyval.location = merge_locations(yyvsp[-1].location,yyvsp[0].location);
2258:     ;}
2259: break;
2260:
2261: case 66:
2262: #line 534 "xTEDS.y"
2263: {
2264:     yyval.location = NULL;
2265:     ;}
2266: break;
2267:
2268: case 67:
2269: #line 540 "xTEDS.y"
2270: {
2271:     location* x_loc = new_location();
2272:     x_loc->x_value = yyvsp[0].str;
2273:     yyval.location = x_loc;
2274:     ;}
2275: break;
2276:
2277: case 68:
2278: #line 546 "xTEDS.y"
2279: {
2280:     location* y_loc = new_location();
2281:     y_loc->y_value = yyvsp[0].str;
2282:     yyval.location = y_loc;
2283:     ;}
2284: break;
2285:
2286: case 69:
2287: #line 552 "xTEDS.y"
2288: {
2289:     location* z_loc = new_location();
2290:     z_loc->z_value = yyvsp[0].str;
2291:     yyval.location = z_loc;
2292:     ;}
2293: break;

```



```

2294:
2295: case 70:
2296: #line 558 "xTEDS.y"
2297: {
2298:     location* units_val = new_location();
2299:     units_val->units = yyvsp[0].str;
2300:     yyval.location = units_val;
2301:     ;}
2302: break;
2303:
2304: case 71:
2305: #line 571 "xTEDS.y"
2306: {
2307:     yyval.orientation = yyvsp[-1].orientation;
2308:     ;}
2309: break;
2310:
2311: case 72:
2312: #line 575 "xTEDS.y"
2313: {
2314:     yyval.orientation = yyvsp[-3].orientation;
2315:     ;}
2316: break;
2317:
2318: case 73:
2319: #line 581 "xTEDS.y"
2320: {
2321:     yyval.orientation = merge_orientations(yyvsp[-1].orientation,yyvsp[0].orientation);
2322:     ;}
2323: break;
2324:
2325: case 74:
2326: #line 585 "xTEDS.y"
2327: {
2328:     yyval.orientation = NULL;
2329:     ;}
2330: break;
2331:
2332: case 75:
2333: #line 591 "xTEDS.y"
2334: {

```

```

2335:         orientation* temp = new_orientation();
2336:         temp->axis_value = yyvsp[0].str;
2337:         yyval.orientation = temp;
2338:     };}
2339:     break;
2340:
2341: case 76:
2342: #line 597 "xTEDS.y"
2343: {
2344:         orientation* temp = new_orientation();
2345:         temp->angle_value = yyvsp[0].str;
2346:         yyval.orientation = temp;
2347:     };}
2348:     break;
2349:
2350: case 77:
2351: #line 603 "xTEDS.y"
2352: {
2353:         orientation* temp = new_orientation();
2354:         temp->units_value = yyvsp[0].str;
2355:         yyval.orientation = temp;
2356:     };}
2357:     break;
2358:
2359: case 78:
2360: #line 615 "xTEDS.y"
2361: {
2362:         yyval.qual= link_qualifiers(yyvsp[-1].qual,yyvsp[0].qual);
2363:     };}
2364:     break;
2365:
2366: case 79:
2367: #line 619 "xTEDS.y"
2368: {
2369:         yyval.qual= NULL;
2370:     };}
2371:     break;
2372:
2373: case 80:
2374: #line 625 "xTEDS.y"
2375: {

```

```

2376:         yyval.qual=yyvsp[-3].qual
2377:     ;}
2378:     break;
2379:
2380: case 81:
2381: #line 629 "xTEDS.y"
2382: {
2383:         yyval.qual=yyvsp[-1].qual
2384:     ;}
2385:     break;
2386:
2387: case 82:
2388: #line 635 "xTEDS.y"
2389: {
2390:         yyval.qual = merge_qualifiers(yyvsp[-1].qual,yyvsp[0].qual);
2391:     ;}
2392:     break;
2393:
2394: case 83:
2395: #line 639 "xTEDS.y"
2396: {
2397:         yyval.qual = NULL;
2398:     ;}
2399:     break;
2400:
2401: case 84:
2402: #line 645 "xTEDS.y"
2403: {
2404:         qualifier_type* temp;
2405:         temp = new_qualifier();
2406:         temp->name = yyvsp[0].str;
2407:         yyval.qual = temp;
2408:     ;}
2409:     break;
2410:
2411: case 85:
2412: #line 652 "xTEDS.y"
2413: {
2414:         qualifier_type* temp;
2415:         temp = new_qualifier();
2416:         temp->value = yyvsp[0].str;

```

```

2417:         yyval.qual = temp;
2418:     ;}
2419: break;
2420:
2421: case 86:
2422: #line 659 "xTEDS.y"
2423: {
2424:         qualifier_type* temp;
2425:         temp = new_qualifier();
2426:         temp->units = yyvsp[0].str;
2427:         yyval.qual = temp;
2428:     ;}
2429: break;
2430:
2431: case 87:
2432: #line 673 "xTEDS.y"
2433: {
2434:         yyval.interface = link_interface(yyvsp[-1].interface,yyvsp[0].interface);
2435:     ;}
2436: break;
2437:
2438: case 88:
2439: #line 677 "xTEDS.y"
2440: {
2441:         yyval.interface = NULL;
2442:     ;}
2443: break;
2444:
2445: case 89:
2446: #line 683 "xTEDS.y"
2447: {
2448:         yyval.interface = interface_add_references(yyvsp[-5].interface,yyvsp[-
4].qual,yyvsp[-3].var,yyvsp[-2].message);
2449:     ;}
2450: break;
2451:
2452: case 90:
2453: #line 687 "xTEDS.y"
2454: {
2455:         yyval.interface = yyvsp[0].interface;
2456:     ;}

```

```

2457: break;
2458:
2459: case 91:
2460: #line 693 "xTEDS.y"
2461: {
2462:         yyval.interface = yyvsp[-1].interface;
2463:     };}
2464: break;
2465:
2466: case 92:
2467: #line 697 "xTEDS.y"
2468: {
2469:         yyval.interface = yyvsp[-1].interface;
2470:     };}
2471: break;
2472:
2473: case 93:
2474: #line 703 "xTEDS.y"
2475: {
2476:         yyval.interface = merge_interface(yyvsp[-1].interface,yyvsp[0].interface);
2477:     };}
2478: break;
2479:
2480: case 94:
2481: #line 707 "xTEDS.y"
2482: {
2483:         interface* temp;
2484:         temp = new_interface();
2485:         yyval.interface = temp;
2486:     };}
2487: break;
2488:
2489: case 95:
2490: #line 715 "xTEDS.y"
2491: {
2492:         /*Only qualifiers will be returned, all others thrown away*/
2493:         yyval.qual = link_qualifiers(yyvsp[-1].qual,yyvsp[0].qual);
2494:     };}
2495: break;
2496:
2497: case 96:

```

```

2498: #line 720 "xTEDS.y"
2499:  {
2500:          yyval.qual = NULL;
2501:      ;}
2502:  break;
2503:
2504: case 97:
2505: #line 726 "xTEDS.y"
2506:  {
2507:          yyval.qual = yyvsp[0].qual;
2508:      ;}
2509:  break;
2510:
2511: case 98:
2512: #line 730 "xTEDS.y"
2513:  {
2514:          /* Don't need to save location information for an interface*/
2515:          delete_location(yyvsp[0].location);
2516:          yyval.qual = NULL;
2517:      ;}
2518:  break;
2519:
2520: case 99:
2521: #line 736 "xTEDS.y"
2522:  {
2523:          /* Don't need to save orientation information for an interface*/
2524:          delete_orientation(yyvsp[0].orientation);
2525:          yyval.qual = NULL;
2526:      ;}
2527:  break;
2528:
2529: case 100:
2530: #line 744 "xTEDS.y"
2531:  {
2532:          interface* temp;
2533:          temp = new_interface();
2534:          temp->name = yyvsp[0].str;
2535:          yyval.interface = temp;
2536:      ;}
2537:  break;
2538:

```

```

2539: case 101:
2540: #line 751 "xTEDS.y"
2541: {
2542:     interface* temp;
2543:     temp = new_interface();
2544:     temp->extends = yyvsp[0].str;
2545:     yyval.interface = temp;
2546:     ;}
2547: break;
2548:
2549: case 102:
2550: #line 758 "xTEDS.y"
2551: {
2552:     interface* temp;
2553:     temp = new_interface();
2554:     temp->id = yyvsp[0].str;
2555:     yyval.interface = temp;
2556:     ;}
2557: break;
2558:
2559: case 103:
2560: #line 765 "xTEDS.y"
2561: {
2562:     interface* temp;
2563:     temp = new_interface();
2564:     temp->description = yyvsp[0].str;
2565:     yyval.interface = temp;
2566:     ;}
2567: break;
2568:
2569: case 104:
2570: #line 779 "xTEDS.y"
2571: {
2572:     yyval.message = merge_message(yyvsp[-1].message,yyvsp[0].message);
2573:     ;}
2574: break;
2575:
2576: case 105:
2577: #line 783 "xTEDS.y"
2578: {
2579:     yyval.message = NULL;

```

```

2580:         ;}
2581:     break;
2582:
2583: case 106:
2584: #line 789 "xTEDS.y"
2585: {
2586:     message* temp;
2587:     temp = new_message();
2588:     temp->commands = yyvsp[0].command;
2589:     yyval.message = temp;
2590:     ;}
2591: break;
2592:
2593: case 107:
2594: #line 796 "xTEDS.y"
2595: {
2596:     message* temp;
2597:     temp = new_message();
2598:     temp->notifications = yyvsp[0].notification;
2599:     yyval.message = temp;
2600:     ;}
2601: break;
2602:
2603: case 108:
2604: #line 803 "xTEDS.y"
2605: {
2606:     message* temp;
2607:     temp = new_message();
2608:     temp->requests = yyvsp[0].request;
2609:     yyval.message = temp;
2610:     ;}
2611: break;
2612:
2613: case 109:
2614: #line 812 "xTEDS.y"
2615: {
2616:     request* temp;
2617:     temp = new_request();
2618:     temp = request_add_cmd_msg(temp,yyvsp[-4].cmd_msg);
2619:     temp = request_add_data_msg(temp,yyvsp[-3].data_msg);
2620:     yyval.request = request_add_fault_msg(temp,yyvsp[-2].fault_msg);

```



```

2621:         ;}
2622:     break;
2623:
2624: case 110:
2625: #line 827 "xTEDS.y"
2626:     {
2627:         yyval.fault_msg = yyvsp[0].fault_msg;
2628:     ;}
2629:     break;
2630:
2631: case 111:
2632: #line 833 "xTEDS.y"
2633:     {
2634:         fault_msg* result;
2635:         result = merge_fault_msg(yyvsp[-5].fault_msg,yyvsp[-3].fault_msg);
2636:         yyval.fault_msg = fault_add_var_refs(result,yyvsp[-2].var_ref);
2637:     ;}
2638:     break;
2639:
2640: case 112:
2641: #line 839 "xTEDS.y"
2642:     {
2643:         yyval.fault_msg = yyvsp[-1].fault_msg;
2644:     ;}
2645:     break;
2646:
2647: case 113:
2648: #line 843 "xTEDS.y"
2649:     {
2650:         yyval.fault_msg = NULL;
2651:     ;}
2652:     break;
2653:
2654: case 114:
2655: #line 849 "xTEDS.y"
2656:     {
2657:         yyval.fault_msg = merge_fault_msg(yyvsp[-1].fault_msg,yyvsp[0].fault_msg);
2658:     ;}
2659:     break;
2660:
2661: case 115:

```

```

2662: #line 853 "xTEDS.y"
2663:  {
2664:          yyval.fault_msg = NULL;
2665:      ;}
2666:  break;
2667:
2668: case 116:
2669: #line 859 "xTEDS.y"
2670:  {
2671:          fault_msg* temp;
2672:          temp = new_fault_msg();
2673:          temp->name = yyvsp[0].str;
2674:          yyval.fault_msg = temp;
2675:      ;}
2676:  break;
2677:
2678: case 117:
2679: #line 866 "xTEDS.y"
2680:  {
2681:          fault_msg* temp;
2682:          temp = new_fault_msg();
2683:          temp->id = yyvsp[0].str;
2684:          yyval.fault_msg = temp;
2685:      ;}
2686:  break;
2687:
2688: case 118:
2689: #line 873 "xTEDS.y"
2690:  {
2691:          fault_msg* temp;
2692:          temp = new_fault_msg();
2693:          temp->description = yyvsp[0].str;
2694:          yyval.fault_msg = temp;
2695:      ;}
2696:  break;
2697:
2698: case 119:
2699: #line 882 "xTEDS.y"
2700:  {
2701:          fault_msg* temp;
2702:          temp = new_fault_msg();

```

```

2703:         temp->qualifiers = yyvsp[0].qual;
2704:         yyval.fault_msg = temp;
2705:     ;}
2706: break;
2707:
2708: case 120:
2709: #line 896 "xTEDS.y"
2710: {
2711:         notification* temp;
2712:         temp = new_notification();
2713:         temp = notification_add_data_msg(temp,yyvsp[-3].data_msg);
2714:         yyval.notification = notification_add_fault_msg(temp,yyvsp[-2].fault_msg);
2715:     ;}
2716: break;
2717:
2718: case 121:
2719: #line 905 "xTEDS.y"
2720: {
2721:         yyval.data_msg = link_data_msg(yyvsp[-1].data_msg,yyvsp[0].data_msg);
2722:     ;}
2723: break;
2724:
2725: case 122:
2726: #line 909 "xTEDS.y"
2727: {
2728:         yyval.data_msg = NULL;
2729:     ;}
2730: break;
2731:
2732: case 123:
2733: #line 915 "xTEDS.y"
2734: {
2735:         data_msg* result;
2736:         result = merge_data_msg(yyvsp[-5].data_msg,yyvsp[-3].data_msg);
2737:         yyval.data_msg = data_add_var_refs(result,yyvsp[-2].var_ref);
2738:     ;}
2739: break;
2740:
2741: case 124:
2742: #line 923 "xTEDS.y"
2743: {

```

```

2744:         yyval.data_msg=merge_data_msg(yyvsp[-1].data_msg,yyvsp[0].data_msg);
2745:     ;}
2746:     break;
2747:
2748: case 125:
2749: #line 927 "xTEDS.y"
2750:     {
2751:         yyval.data_msg=NULL;
2752:     ;}
2753:     break;
2754:
2755: case 126:
2756: #line 933 "xTEDS.y"
2757:     {
2758:         data_msg* temp;
2759:         temp = new_data_msg();
2760:         temp->name = yyvsp[0].str;
2761:         yyval.data_msg = temp;
2762:     ;}
2763:     break;
2764:
2765: case 127:
2766: #line 940 "xTEDS.y"
2767:     {
2768:         data_msg* temp;
2769:         temp = new_data_msg();
2770:         temp->id = yyvsp[0].str;
2771:         yyval.data_msg = temp;
2772:     ;}
2773:     break;
2774:
2775: case 128:
2776: #line 947 "xTEDS.y"
2777:     {
2778:         data_msg* temp;
2779:         temp = new_data_msg();
2780:         temp->msg_arrival = yyvsp[0].str;
2781:         yyval.data_msg = temp;
2782:     ;}
2783:     break;
2784:

```

```

2785: case 129:
2786: #line 954 "xTEDS.y"
2787: {
2788:     data_msg* temp;
2789:     temp = new_data_msg();
2790:     temp->description = yyvsp[0].str;
2791:     yyval.data_msg = temp;
2792:     ;}
2793: break;
2794:
2795: case 130:
2796: #line 961 "xTEDS.y"
2797: {
2798:     data_msg* temp;
2799:     temp = new_data_msg();
2800:     temp->msg_rate = yyvsp[0].str;
2801:     yyval.data_msg = temp;
2802:     ;}
2803: break;
2804:
2805: case 131:
2806: #line 970 "xTEDS.y"
2807: {
2808:     data_msg* temp;
2809:     temp = new_data_msg();
2810:     temp->qualifiers = yyvsp[0].qual;
2811:     yyval.data_msg = temp;
2812:     ;}
2813: break;
2814:
2815: case 132:
2816: #line 984 "xTEDS.y"
2817: {
2818:     command* temp;
2819:     temp = new_command();
2820:     temp = command_add_cmd_msg(temp,yyvsp[-3].cmd_msg);
2821:     yyval.command = command_add_fault_msg(temp,yyvsp[-2].fault_msg);
2822:     ;}
2823: break;
2824:
2825: case 133:

```

```

2826: #line 993 "xTEDS.y"
2827:  {
2828:          yyval.cmd_msg = link_cmd_msg(yyvsp[-1].cmd_msg,yyvsp[0].cmd_msg);
2829:      ;}
2830:  break;
2831:
2832: case 134:
2833: #line 997 "xTEDS.y"
2834:  {
2835:          yyval.cmd_msg = NULL;
2836:      ;}
2837:  break;
2838:
2839: case 135:
2840: #line 1003 "xTEDS.y"
2841:  {
2842:          cmd_msg* result;
2843:          result = merge_cmd_msg(yyvsp[-5].cmd_msg,yyvsp[-3].cmd_msg);
2844:          yyval.cmd_msg = cmd_add_var_refs(result,yyvsp[-2].var_ref);
2845:      ;}
2846:  break;
2847:
2848: case 136:
2849: #line 1009 "xTEDS.y"
2850:  {
2851:          yyval.cmd_msg = cmd_add_var_refs(yyvsp[-1].cmd_msg,NULL);
2852:      ;}
2853:  break;
2854:
2855: case 137:
2856: #line 1015 "xTEDS.y"
2857:  {
2858:          yyval.cmd_msg = merge_cmd_msg(yyvsp[-1].cmd_msg,yyvsp[0].cmd_msg);
2859:      ;}
2860:  break;
2861:
2862: case 138:
2863: #line 1019 "xTEDS.y"
2864:  {
2865:          yyval.cmd_msg=NULL;
2866:      ;}

```

```

2867: break;
2868:
2869: case 139:
2870: #line 1025 "xTEDS.y"
2871: {
2872:     cmd_msg* temp;
2873:     temp = new_cmd_msg();
2874:     temp->name = yyvsp[0].str;
2875:     yyval.cmd_msg = temp;
2876:     ;}
2877: break;
2878:
2879: case 140:
2880: #line 1032 "xTEDS.y"
2881: {
2882:     cmd_msg* temp;
2883:     temp = new_cmd_msg();
2884:     temp->id = yyvsp[0].str;
2885:     yyval.cmd_msg = temp;
2886:     ;}
2887: break;
2888:
2889: case 141:
2890: #line 1039 "xTEDS.y"
2891: {
2892:     cmd_msg* temp;
2893:     temp = new_cmd_msg();
2894:     temp->description = yyvsp[0].str;
2895:     yyval.cmd_msg = temp;
2896:     ;}
2897: break;
2898:
2899: case 142:
2900: #line 1048 "xTEDS.y"
2901: {
2902:     cmd_msg* temp;
2903:     temp = new_cmd_msg();
2904:     temp->qualifiers = yyvsp[0].qual;
2905:     yyval.cmd_msg = temp;
2906:     ;}
2907: break;

```

```

2908:
2909: case 143:
2910: #line 1062 "xTEDS.y"
2911: {
2912:     yyval.var= link_variables(yyvsp[-1].var,yyvsp[0].var);
2913:     ;}
2914: break;
2915:
2916: case 144:
2917: #line 1066 "xTEDS.y"
2918: {
2919:     yyval.var= NULL;
2920:     ;}
2921: break;
2922:
2923: case 145:
2924: #line 1072 "xTEDS.y"
2925: {
2926:     yyval.var=yyvsp[0].var
2927:     ;}
2928: break;
2929:
2930: case 146:
2931: #line 1076 "xTEDS.y"
2932: {
2933:     yyval.var=yyvsp[0].var
2934:     ;}
2935: break;
2936:
2937: case 147:
2938: #line 1082 "xTEDS.y"
2939: {
2940:     yyval.var = merge_variables(yyvsp[-4].var,yyvsp[-2].var);
2941:     ;}
2942: break;
2943:
2944: case 149:
2945: #line 1091 "xTEDS.y"
2946: {
2947:     yyval.var= yyvsp[0].var
2948:     ;}

```



```

2949: break;
2950:
2951: case 150:
2952: #line 1097 "xTEDS.y"
2953: {
2954:     yyval.var= merge_variables(yyvsp[-1].var,yyvsp[0].var);
2955:     ;}
2956: break;
2957:
2958: case 151:
2959: #line 1101 "xTEDS.y"
2960: {
2961:     yyval.var=NULL;
2962:     ;}
2963: break;
2964:
2965: case 152:
2966: #line 1107 "xTEDS.y"
2967: {
2968:     variable* temp;
2969:     temp = new_variable();
2970:     temp->name = yyvsp[0].str;
2971:     yyval.var = temp;
2972:     ;}
2973: break;
2974:
2975: case 153:
2976: #line 1114 "xTEDS.y"
2977: {
2978:     variable* temp;
2979:     temp = new_variable();
2980:     temp->kind = yyvsp[0].str;
2981:     yyval.var = temp;
2982:     ;}
2983: break;
2984:
2985: case 154:
2986: #line 1121 "xTEDS.y"
2987: {
2988:     variable* temp;
2989:     temp = new_variable();

```

```

2990:         temp->format = yyvsp[0].str;
2991:         yyval.var = temp;
2992:     ;}
2993: break;
2994:
2995: case 155:
2996: #line 1128 "xTEDS.y"
2997: {
2998:     variable* temp;
2999:     temp = new_variable();
3000:     temp->qualifier = yyvsp[0].str;
3001:     printf("<Variable qualifier attribute has been deprecated! \n");
3002:     yyval.var = temp;
3003: ;}
3004: break;
3005:
3006: case 156:
3007: #line 1136 "xTEDS.y"
3008: {
3009:     variable* temp;
3010:     temp = new_variable();
3011:     temp->id = yyvsp[0].str;
3012:     yyval.var = temp;
3013: ;}
3014: break;
3015:
3016: case 157:
3017: #line 1143 "xTEDS.y"
3018: {
3019:     variable* temp;
3020:     temp = new_variable();
3021:     temp->description = yyvsp[0].str;
3022:     yyval.var = temp;
3023: ;}
3024: break;
3025:
3026: case 158:
3027: #line 1150 "xTEDS.y"
3028: {
3029:     variable* temp;
3030:     temp = new_variable();

```

```

3031:         temp->range_min = yyvsp[0].str;
3032:         yyval.var = temp;
3033:     ;}
3034: break;
3035:
3036: case 159:
3037: #line 1157 "xTEDS.y"
3038: {
3039:     variable* temp;
3040:     temp = new_variable();
3041:     temp->range_max = yyvsp[0].str;
3042:     yyval.var = temp;
3043: ;}
3044: break;
3045:
3046: case 160:
3047: #line 1164 "xTEDS.y"
3048: {
3049:     variable* temp;
3050:     temp = new_variable();
3051:     temp->length = yyvsp[0].str;
3052:     yyval.var = temp;
3053: ;}
3054: break;
3055:
3056: case 161:
3057: #line 1171 "xTEDS.y"
3058: {
3059:     variable* temp;
3060:     temp = new_variable();
3061:     temp->default_value = yyvsp[0].str;
3062:     yyval.var = temp;
3063: ;}
3064: break;
3065:
3066: case 162:
3067: #line 1178 "xTEDS.y"
3068: {
3069:     variable* temp;
3070:     temp = new_variable();
3071:     temp->precision = yyvsp[0].str;

```

```

3072:         yyval.var = temp;
3073:     ;}
3074: break;
3075:
3076: case 163:
3077: #line 1185 "xTEDS.y"
3078: {
3079:     variable* temp;
3080:     temp = new_variable();
3081:     temp->units = yyvsp[0].str;
3082:     yyval.var = temp;
3083: ;}
3084: break;
3085:
3086: case 164:
3087: #line 1192 "xTEDS.y"
3088: {
3089:     variable* temp;
3090:     temp = new_variable();
3091:     temp->accuracy = yyvsp[0].str;
3092:     yyval.var = temp;
3093: ;}
3094: break;
3095:
3096: case 165:
3097: #line 1199 "xTEDS.y"
3098: {
3099:     variable* temp;
3100:     temp = new_variable();
3101:     temp->scale_factor = yyvsp[0].str;
3102:     yyval.var = temp;
3103: ;}
3104: break;
3105:
3106: case 166:
3107: #line 1206 "xTEDS.y"
3108: {
3109:     variable* temp;
3110:     temp = new_variable();
3111:     temp->scale_units = yyvsp[0].str;
3112:     yyval.var = temp;

```

```

3113:         ;}
3114:     break;
3115:
3116: case 167:
3117: #line 1213 "xTEDS.y"
3118:     {
3119:         variable* temp;
3120:         temp = new_variable();
3121:         temp->r_low = yyvsp[0].str;
3122:         yyval.var = temp;
3123:     ;}
3124:     break;
3125:
3126: case 168:
3127: #line 1220 "xTEDS.y"
3128:     {
3129:         variable* temp;
3130:         temp = new_variable();
3131:         temp->r_high = yyvsp[0].str;
3132:         yyval.var = temp;
3133:     ;}
3134:     break;
3135:
3136: case 169:
3137: #line 1227 "xTEDS.y"
3138:     {
3139:         variable* temp;
3140:         temp = new_variable();
3141:         temp->y_low = yyvsp[0].str;
3142:         yyval.var = temp;
3143:     ;}
3144:     break;
3145:
3146: case 170:
3147: #line 1234 "xTEDS.y"
3148:     {
3149:         variable* temp;
3150:         temp = new_variable();
3151:         temp->y_high = yyvsp[0].str;
3152:         yyval.var = temp;
3153:     ;}

```

```

3154: break;
3155:
3156: case 171:
3157: #line 1242 "xTEDS.y"
3158: {
3159:     variable* temp;
3160:     temp = new_variable();
3161:     temp->invalid_value = yyvsp[0].str;
3162:     yyval.var = temp;
3163:     ;}
3164: break;
3165:
3166: case 172:
3167: #line 1251 "xTEDS.y"
3168: {
3169:     yyval.var = merge_variables(yyvsp[-1].var,yyvsp[0].var);
3170:     ;}
3171: break;
3172:
3173: case 173:
3174: #line 1257 "xTEDS.y"
3175: {
3176:     variable* temp;
3177:     temp = new_variable();
3178:     temp->qualifiers = yyvsp[0].qual;
3179:     yyval.var = temp;
3180:     ;}
3181: break;
3182:
3183: case 174:
3184: #line 1267 "xTEDS.y"
3185: {
3186:     yyval.var = merge_variables(yyvsp[-1].var,yyvsp[0].var);
3187:     ;}
3188: break;
3189:
3190: case 175:
3191: #line 1271 "xTEDS.y"
3192: {
3193:     yyval.var = NULL;
3194:     ;}

```

```

3195: break;
3196:
3197: case 176:
3198: #line 1277 "xTEDS.y"
3199: {
3200:     variable* temp;
3201:     temp = new_variable();
3202:     temp->dranges = yyvsp[0].drange;
3203:     yyval.var = temp;
3204:     ;}
3205: break;
3206:
3207: case 177:
3208: #line 1284 "xTEDS.y"
3209: {
3210:     variable* temp;
3211:     temp = new_variable();
3212:     temp->curves = yyvsp[0].curve;
3213:     yyval.var = temp;
3214:     ;}
3215: break;
3216:
3217: case 178:
3218: #line 1291 "xTEDS.y"
3219: {
3220:     variable* temp;
3221:     temp = new_variable();
3222:     temp->location_data = yyvsp[0].location;
3223:     yyval.var = temp;
3224:     ;}
3225: break;
3226:
3227: case 179:
3228: #line 1298 "xTEDS.y"
3229: {
3230:     variable* temp;
3231:     temp = new_variable();
3232:     temp->orientation_data = yyvsp[0].orientation;
3233:     yyval.var = temp;
3234:     ;}
3235: break;

```

```

3236:
3237: case 180:
3238: #line 1312 "xTEDS.y"
3239: {
3240:     yyval.var_ref = link_variable_ref(yyvsp[-1].var_ref,yyvsp[0].var_ref);
3241:     ;}
3242: break;
3243:
3244: case 181:
3245: #line 1316 "xTEDS.y"
3246: {
3247:     yyval.var_ref = NULL;
3248:     ;}
3249: break;
3250:
3251: case 182:
3252: #line 1322 "xTEDS.y"
3253: {
3254:     yyval.var_ref = new_variable_ref(yyvsp[-1].str);
3255:     ;}
3256: break;
3257:
3258: case 183:
3259: #line 1326 "xTEDS.y"
3260: {
3261:     yyval.var_ref = new_variable_ref(yyvsp[-3].str);
3262:     ;}
3263: break;
3264:
3265: case 184:
3266: #line 1337 "xTEDS.y"
3267: {
3268:     drange* temp;
3269:     temp = new_drange();
3270:     temp->options = yyvsp[-2].curveoption;
3271:     yyval.drange = merge_dranges(yyvsp[-3].drange,temp);
3272:     ;}
3273: break;
3274:
3275: case 185:
3276: #line 1346 "xTEDS.y"

```



```

3277:  {
3278:          yyval.drange = yyvsp[-1].drange;
3279:      ;}
3280:  break;
3281:
3282:  case 186:
3283: #line 1352 "xTEDS.y"
3284:  {
3285:          yyval.drange = merge_dranges(yyvsp[-1].drange,yyvsp[0].drange);
3286:      ;}
3287:  break;
3288:
3289:  case 187:
3290: #line 1356 "xTEDS.y"
3291:  {
3292:          yyval.drange = NULL;
3293:      ;}
3294:  break;
3295:
3296:  case 188:
3297: #line 1362 "xTEDS.y"
3298:  {
3299:          drange* temp;
3300:          temp = new_drange();
3301:          temp->name = yyvsp[0].str;
3302:          yyval.drange = temp;
3303:      ;}
3304:  break;
3305:
3306:  case 189:
3307: #line 1369 "xTEDS.y"
3308:  {
3309:          drange* temp;
3310:          temp = new_drange();
3311:          temp->description = yyvsp[0].str;
3312:          yyval.drange = temp;
3313:      ;}
3314:  break;
3315:
3316:  case 190:
3317: #line 1378 "xTEDS.y"

```

```

3318:  {
3319:          yyval.curveoption = link_options(yyvsp[-1].curveoption,yyvsp[0].curveoption);
3320:      ;}
3321:  break;
3322:
3323:  case 191:
3324: #line 1382 "xTEDS.y"
3325:  {
3326:          yyval.curveoption = NULL;
3327:      ;}
3328:  break;
3329:
3330:  case 192:
3331: #line 1388 "xTEDS.y"
3332:  {
3333:          yyval.curveoption = yyvsp[-1].curveoption;
3334:      ;}
3335:  break;
3336:
3337:  case 193:
3338: #line 1392 "xTEDS.y"
3339:  {
3340:          yyval.curveoption = yyvsp[-3].curveoption;
3341:      ;}
3342:  break;
3343:
3344:  case 194:
3345: #line 1398 "xTEDS.y"
3346:  {
3347:          yyval.curveoption = merge_options(yyvsp[-1].curveoption,yyvsp[0].curveoption);
3348:      ;}
3349:  break;
3350:
3351:  case 195:
3352: #line 1402 "xTEDS.y"
3353:  {
3354:          yyval.curveoption = NULL;
3355:      ;}
3356:  break;
3357:
3358:  case 196:

```

```

3359: #line 1408 "xTEDS.y"
3360:  {
3361:      curveoption* temp;
3362:      temp = new_option();
3363:      temp->name = yyvsp[0].str;
3364:      yyval.curveoption = temp;
3365:      ;}
3366:  break;
3367:
3368:  case 197:
3369: #line 1415 "xTEDS.y"
3370:  {
3371:      curveoption* temp;
3372:      temp = new_option();
3373:      temp->value = yyvsp[0].str;
3374:      yyval.curveoption = temp;
3375:      ;}
3376:  break;
3377:
3378:  case 198:
3379: #line 1422 "xTEDS.y"
3380:  {
3381:      curveoption* temp;
3382:      temp = new_option();
3383:      temp->description = yyvsp[0].str;
3384:      yyval.curveoption = temp;
3385:      ;}
3386:  break;
3387:
3388:  case 199:
3389: #line 1429 "xTEDS.y"
3390:  {
3391:      curveoption* temp;
3392:      temp = new_option();
3393:      temp->alarm = yyvsp[0].str;
3394:      yyval.curveoption = temp;
3395:      ;}
3396:  break;
3397:
3398:  case 200:
3399: #line 1443 "xTEDS.y"

```

```

3400:  {
3401:      curve* temp;
3402:      temp = new_curve();
3403:      temp->coefs = yyvsp[-2].coef;
3404:      yyval.curve = merge_curves(yyvsp[-3].curve,temp);
3405:      ;}
3406:  break;
3407:
3408:  case 201:
3409: #line 1452 "xTEDS.y"
3410:  {
3411:      yyval.curve = yyvsp[-1].curve;
3412:      ;}
3413:  break;
3414:
3415:  case 202:
3416: #line 1458 "xTEDS.y"
3417:  {
3418:      yyval.curve = merge_curves(yyvsp[-1].curve,yyvsp[0].curve);
3419:      ;}
3420:  break;
3421:
3422:  case 203:
3423: #line 1462 "xTEDS.y"
3424:  {
3425:      yyval.curve = NULL;
3426:      ;}
3427:  break;
3428:
3429:  case 204:
3430: #line 1468 "xTEDS.y"
3431:  {
3432:      curve* temp;
3433:      temp = new_curve();
3434:      temp->name = yyvsp[0].str;
3435:      yyval.curve = temp;
3436:      ;}
3437:  break;
3438:
3439:  case 205:
3440: #line 1475 "xTEDS.y"

```

```

3441:  {
3442:      curve* temp;
3443:      temp = new_curve();
3444:      temp->description = yyvsp[0].str;
3445:      yyval.curve = temp;
3446:  };}
3447:  break;
3448:
3449: case 206:
3450: #line 1484 "xTEDS.y"
3451:  {
3452:      yyval.coef = link_coefs(yyvsp[-1].coef,yyvsp[0].coef);
3453:  };}
3454:  break;
3455:
3456: case 207:
3457: #line 1488 "xTEDS.y"
3458:  {
3459:      yyval.coef = NULL;
3460:  };}
3461:  break;
3462:
3463: case 208:
3464: #line 1494 "xTEDS.y"
3465:  {
3466:      yyval.coef = yyvsp[-1].coef;
3467:  };}
3468:  break;
3469:
3470: case 209:
3471: #line 1498 "xTEDS.y"
3472:  {
3473:      yyval.coef = yyvsp[-3].coef;
3474:  };}
3475:  break;
3476:
3477: case 210:
3478: #line 1502 "xTEDS.y"
3479:  {
3480:      yyval.coef = yyvsp[-1].coef;
3481:  };}

```

```

3482: break;
3483:
3484: case 211:
3485: #line 1508 "xTEDS.y"
3486: {
3487:     yyval.coef = merge_coefs(yyvsp[-1].coef,yyvsp[0].coef);
3488:     ;}
3489: break;
3490:
3491: case 212:
3492: #line 1512 "xTEDS.y"
3493: {
3494:     yyval.coef = NULL;
3495:     ;}
3496: break;
3497:
3498: case 213:
3499: #line 1518 "xTEDS.y"
3500: {
3501:     coef* temp;
3502:     temp = new_coef();
3503:     temp->exponent = yyvsp[0].str;
3504:     yyval.coef = temp;
3505:     ;}
3506: break;
3507:
3508: case 214:
3509: #line 1525 "xTEDS.y"
3510: {
3511:     coef* temp;
3512:     temp = new_coef();
3513:     temp->value = yyvsp[0].str;
3514:     yyval.coef = temp;
3515:     ;}
3516: break;
3517:
3518: case 215:
3519: #line 1532 "xTEDS.y"
3520: {
3521:     coef* temp;
3522:     temp = new_coef();

```

```

3523:         temp->description = yyvsp[0].str;
3524:         yyval.coef = temp;
3525:     };}
3526: break;
3527:
3528:
3529: }
3530:
3531: /* Line 1010 of yacc.c. */
3532: #line 3533 "xTEDS.tab.c"
3533:
3534: yyvsp -= yylen;
3535: yyssp -= yylen;
3536:
3537:
3538: YY_STACK_PRINT (yyss, yyssp);
3539:
3540: *++yyvsp = yyval;
3541:
3542:
3543: /* Now `shift' the result of the reduction.  Determine what state
3544:    that goes to, based on the state we popped back to and the rule
3545:    number reduced by.  */
3546:
3547: yyn = yyr1[yyn];
3548:
3549: yystate = yypgoto[yyn - YYNTOKENS] + *yyssp;
3550: if (0 <= yystate && yystate <= YYLAST && yycheck[yystate] == *yyssp)
3551:     yystate = yytable[yystate];
3552: else
3553:     yystate = yydefgoto[yyn - YYNTOKENS];
3554:
3555: goto yynewstate;
3556:
3557:
3558: /*-----,
3559: | yyerrlab -- here on detecting error |
3560: `-----*/
3561: yyerrlab:
3562: /* If not already recovering from an error, report this error.  */
3563: if (!yyerrstatus)

```

```

3564: {
3565:     ++yynerrs;
3566: #if YYERROR_VERBOSE
3567:     yyn = yypact[yystate];
3568:
3569:     if (YYPACT_NINF < yyn && yyn < YYLAST)
3570:     {
3571:         YYSIZE_T yysize = 0;
3572:         int yytype = YYTRANSLATE (yychar);
3573:         const char* yyprefix;
3574:         char *yymsg;
3575:         int yyx;
3576:
3577:         /* Start YYX at -YYN if negative to avoid negative indexes in
3578:            YYCHECK. */
3579:         int yyxbegin = yyn < 0 ? -yyn : 0;
3580:
3581:         /* Stay within bounds of both yycheck and yytname. */
3582:         int yychecklim = YYLAST - yyn;
3583:         int yyxend = yychecklim < YYNTOKENS ? yychecklim : YYNTOKENS;
3584:         int yycount = 0;
3585:
3586:         yyprefix = ", expecting ";
3587:         for (yyx = yyxbegin; yyx < yyxend; ++yyx)
3588:             if (yycheck[yyx + yyn] == yyx && yyx != YYTERROR)
3589:             {
3590:                 yysize += yystrlen (yyprefix) + yystrlen (yytname [yyx]);
3591:                 yycount += 1;
3592:                 if (yycount == 5)
3593:                 {
3594:                     yysize = 0;
3595:                     break;
3596:                 }
3597:             }
3598:         yysize += (sizeof ("syntax error, unexpected ")
3599:                    + yystrlen (yytname[yytype]));
3600:         yymsg = (char *) YYSTACK_ALLOC (yysize);
3601:         if (yymsg != 0)
3602:         {
3603:             char *yyp = yystpcpy (yymsg, "syntax error, unexpected ");
3604:             yyp = yystpcpy (yyp, yytname[yytype]);

```



```

3605:
3606:     if (yycount < 5)
3607:     {
3608:         yyprefix = ", expecting ";
3609:         for (yyx = yyxbegin; yyx < yyxend; ++yyx)
3610:             if (yycheck[yyx + yyn] == yyx && yyx != YYERROR)
3611:             {
3612:                 yyp = yystpcpy (yyp, yyprefix);
3613:                 yyp = yystpcpy (yyp, yytname[yyx]);
3614:                 yyprefix = " or ";
3615:             }
3616:     }
3617:     yyerror (yymsg);
3618:     YYSTACK_FREE (yymsg);
3619: }
3620: else
3621:     yyerror ("syntax error; also virtual memory exhausted");
3622: }
3623: else
3624: #endif /* YYERROR_VERBOSE */
3625: yyerror ("syntax error");
3626: }
3627:
3628:
3629:
3630: if (yyerrstatus == 3)
3631: {
3632:     /* If just tried and failed to reuse lookahead token after an
3633: error, discard it. */
3634:
3635:     if (yychar <= YYEOF)
3636:     {
3637:         /* If at end of input, pop the error token,
3638: then the rest of the stack, then return failure. */
3639:         if (yychar == YYEOF)
3640:             for (;;)
3641:             {
3642:                 YYPOPSTACK;
3643:                 if (yyssp == yyss)
3644:                     YYABORT;
3645:                 YYDSYMPRINTF ("Error: popping", yystos[*yyssp], yyvsp, yylsp);

```

```

3646:     yydestruct (yystos[*yyssp], yyvsp);
3647: }
3648: }
3649: else
3650: {
3651:     YYDSYMPRINTF ("Error: discarding", yytoken, &yylval, &yyloc);
3652:     yydestruct (yytoken, &yylval);
3653:     yychar = YYEMPTY;
3654:
3655: }
3656: }
3657:
3658: /* Else will try to reuse lookahead token after shifting the error
3659:    token. */
3660: goto yyerrlab1;
3661:
3662:
3663: /*-----
3664: | yyerrorlab -- error raised explicitly by YYERROR. |
3665: `-----*/
3666: yyerrorlab:
3667:
3668: #ifdef __GNUC__
3669: /* Pacify GCC when the user code never invokes YYERROR and the label
3670:    yyerrorlab therefore never appears in user code. */
3671: if (0)
3672:     goto yyerrorlab;
3673: #endif
3674:
3675: yyvsp -= yylen;
3676: yyssp -= yylen;
3677: yystate = *yyssp;
3678: goto yyerrlab1;
3679:
3680:
3681: /*-----
3682: | yyerrlab1 -- common code for both syntax error and YYERROR. |
3683: `-----*/
3684: yyerrlab1:
3685:     yyerrstatus = 3; /* Each real token shifted decrements this. */
3686:

```

```

3687: for (;;)
3688: {
3689:     yyn = yypact[yystate];
3690:     if (yyn != YYPACT_NINF)
3691:     {
3692:         yyn += YYERROR;
3693:         if (0 <= yyn && yyn <= YYLAST && yycheck[yyn] == YYERROR)
3694:         {
3695:             yyn = yytable[yyn];
3696:             if (0 < yyn)
3697:                 break;
3698:         }
3699:     }
3700:
3701:     /* Pop the current state because it cannot handle the error token. */
3702:     if (yyssp == yyss)
3703:         YYABORT;
3704:
3705:     YYDSYMPRINTF ("Error: popping", yystos[*yyssp], yyvsp, yylsp);
3706:     yydestruct (yystos[yystate], yyvsp);
3707:     YYPOPSTACK;
3708:     yystate = *yyssp;
3709:     YY_STACK_PRINT (yyss, yyssp);
3710: }
3711:
3712: if (yyn == YYFINAL)
3713:     YYACCEPT;
3714:
3715: YYDPRINTF ((stderr, "Shifting error token, "));
3716:
3717: *++yyvsp = yylval;
3718:
3719:
3720: yystate = yyn;
3721: goto yynewstate;
3722:
3723:
3724: /*-----
3725: | yyacceptlab -- YYACCEPT comes here. |
3726: `-----*/
3727: yyacceptlab:

```

```

3728: yyresult = 0;
3729: goto yyreturn;
3730:
3731: /*-----
3732: | yyabortlab -- YYABORT comes here. |
3733: `-----*/
3734: yyabortlab:
3735: yyresult = 1;
3736: goto yyreturn;
3737:
3738: #ifndef yyoverflow
3739: /*-----
3740: | yyoverflowlab -- parser overflow comes here. |
3741: `-----*/
3742: yyoverflowlab:
3743: yyerror ("parser stack overflow");
3744: yyresult = 2;
3745: /* Fall through. */
3746: #endif
3747:
3748: yyreturn:
3749: #ifndef yyoverflow
3750: if (yyss != yyssa)
3751:   YYSTACK_FREE (yyss);
3752: #endif
3753: return yyresult;
3754: }
3755:
3756:
3757: #line 1539 "xTEDS.y"
3758:
3759:

```

File: sdm/common/xTEDS/xTEDSRequest.h

```
1: #ifndef __SDM_XTEDS_REQUEST_H_
2: #define __SDM_XTEDS_REQUEST_H_
3:
4: #include "xTEDSItem.h"
5: #include "xTEDSDataMsg.h"
6: #include "xTEDSCommandMsg.h"
7: #include "xTEDSFaultMsg.h"
8: #include "xTEDSWrapper.h"
9: #include "xTEDSVariableList.h"
10:
11: class xTEDSRequest:public xTEDSWrapper
12: {
13: public:
14: xTEDSRequest();
15: xTEDSRequest(const xTEDSRequest&);
16: virtual ~xTEDSRequest();
17:
18: virtual MessageDef* RegInfo(const char* ItemName) const;
19: virtual bool RegInfoMatch(const char* Name, const xTEDSQualifierList&, const char* Interface)
const;
20: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
Interface) const;
21: bool SetRequest(const request* Request, const xTEDSVariableList& VariablesList);
22: bool ContainsMessage(const SDMMMessage_ID& MessageID) const;
23: SDMMMessage_ID GetFaultMessageID() const;
24: SDMMMessage_ID GetDataMessageID() const;
25: xTEDSRequest& operator=(const xTEDSRequest&);
26: virtual void PrintDebug() const;
27: private:
28: xTEDSDataMsg* m_ReplyMessage;
29: xTEDSCommandMsg* m_CommandMessage;
30: xTEDSFaultMsg* m_FaultMessage;
31: };
32:
33: #endif
```

File: sdm/common/xTEDS/xTEDSWrapper.h

```
1: #ifndef _SDM_XTEDS_WRAPPER_H_
2: #define _SDM_XTEDS_WRAPPER_H_
3:
4: #include <stdlib.h>
5: #include "MessageDef.h"
6: #include "xTEDSQualifierList.h"
7: #include "../message/SDMMessage_ID.h"
8:
9: class SDMLIB_API xTEDSWrapper
10: {
11: public:
12:     enum WrapperType { WRAPPER_NOTIFICATION, WRAPPER_REQUEST,
WRAPPER_COMMAND, WRAPPER_EMPTY };
13:
14: xTEDSWrapper();
15: xTEDSWrapper(const xTEDSWrapper&);
16: xTEDSWrapper& operator=(const xTEDSWrapper&);
17: virtual ~xTEDSWrapper();
18:
19: virtual MessageDef* RegInfo(const char* ItemName) const = 0;
20: virtual bool RegInfoMatch(const char* Name, const xTEDSQualifierList& Qualifiers, const char*
Interface) const = 0;
21: virtual bool RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const char*
Interface) const = 0;
22: virtual bool ContainsMessage(const SDMMessage_ID& MessageID) const = 0;
23: virtual SDMMessage_ID GetFaultMessageID() const = 0;
24: void setInterfaceName(const char* Name);
25: void setInterfaceID(int ID);
26: WrapperType GetType() const { return Type; }
27: virtual void PrintDebug() const;
28: protected:
29: WrapperType Type;
30: char* InterfaceName;
31: int InterfaceID;
32: };
33:
34: #endif
35:
```

File: sdm/common/xTEDS/xTEDSVerification.h

```
1: #ifndef _SDM_XTEDS_VERIFICATION_H_
2: #define _SDM_XTEDS_VERIFICATION_H_
3:
4: #include <stdlib.h>
5: #include <stdio.h>
6: #include <string.h>
7: extern "C"
8: {
9: #include "xTEDSParser.h"
10: }
11:
12: class SDMLIB_API xTEDSVerification
13: {
14: public:
15: static bool VerifyQualifiers(const char* ItemName, const char* Interface, qualifier_type*
QualifierList);
16: static bool VerifyOrientation(const variable* var);
17: static bool VerifyLocation(const variable* var, const location* loc);
18: static bool VerifyCurve (const variable* var);
19: static bool VerifyDrange (const variable* var);
20: static bool VerifyVarRefs(const char* MsgName, var_ref* VarRefs);
21: static bool VerifyVariable(variable* var);
22: static bool VerifyDataMsg(const data_msg* dat);
23: static bool VerifyCommandMsg(const cmd_msg* cmd);
24: static bool VerifyFaultMsg(const fault_msg* fault);
25: static bool VerifyInterface(const interface* Interface);
26: };
27:
28:
29: #endif
30:
```

File: sdm/common/xTEDS/SDMDataTypes.h

```
1: #ifndef __SDM_DATA_TYPES_H_
2: #define __SDM_DATA_TYPES_H_
3:
4:
5:
6: enum SDMDataTypes
7: {
8:  SDM_UINT08,
9:  SDM_INT08,
10: SDM_UINT16,
11: SDM_INT16,
12: SDM_UINT32,
13: SDM_INT32,
14: SDM_FLOAT32,
15: SDM_FLOAT64
16: };
17:
18: #endif
```


File: sdm/common/xTEDS/VariableDef.cpp

```
1: #include "VariableDef.h"
2:
3: #include <stdlib.h>
4: #include <stdio.h>
5: #include <string.h>
6:
7:                                     VariableDef::VariableDef():m_pDef(NULL),
m_strInterfaceName(),m_strVariableName(),next(NULL)
8: {
9:   m_strInterfaceName[0] = m_strVariableName[0] = '\0';
10: }
11:
12:                                     //VariableDef::VariableDef(const
                                     VariableDef&
b):def(b.def),interface_name(b.interface_name),next(b.next)
13: //{
14:
15: VariableDef::~~VariableDef()
16: {
17:   if(m_pDef != NULL)
18:   {
19:     free(m_pDef);
20:   }
21:   if(next != NULL)
22:   {
23:     delete next;
24:   }
25: }
26:
27: //VariableDef& VariableDef::operator=(const VariableDef& b)
28: //{
29: //   def = b.def;
30: //   interface_name = b.interface_name;
31: //   next = b.next;
32: //   return *this;
33: //}
34:
35: void VariableDef::Join(VariableDef* b)
36: {
37: //join the two lists
```

```

38: VariableDef* cur;
39: if(b == NULL)
40:     return;
41: if(next == NULL)
42: {
43:     next = b;
44:     return;
45: }
46: for(cur = next;cur->next!=NULL;cur=cur->next);
47: cur->next = b;
48: }
49:
50: void VariableDef::SetDefinitions(const char* strNewDefinitions)
51: {
52:     if (m_pDef != NULL)
53:         free (m_pDef);
54:
55:     m_pDef = strdup(strNewDefinitions);
56: }
57:
58: void VariableDef::SetInterfaceName(const char* strNewInterfaceName)
59: {
60:     strncpy(m_strInterfaceName, strNewInterfaceName, sizeof(m_strInterfaceName));
61:     m_strInterfaceName[sizeof(m_strInterfaceName) - 1] = '\0';
62: }
63:
64: void VariableDef::SetVariableName(const char* strNewVariableName)
65: {
66:     strncpy(m_strVariableName, strNewVariableName, sizeof(m_strVariableName));
67:     m_strVariableName[sizeof(m_strVariableName) - 1] = '\0';
68: }
69:
70: void VariableDef::ToStringConcatVariableDefs(char* pReturnBuffer, unsigned int uiBufferSize,
71:     VariableDef* pDefs1, VariableDef* pDefs2, bool bIgnoreDuplicates)
72: {
73:     unsigned int uiCurSize = 0;
74:     unsigned int uiCurLength = 0;
75:
76:     // Either can be Null by itself, but if both null, quit
77:     if (pDefs1 == NULL && pDefs2 == NULL)
78:         return;

```

```

79:
80: for (VariableDef* pCur = pDefs1; pCur != NULL; pCur = pCur->next)
81: {
82:     const char* pCurDefinitions = pCur->GetDefinitions();
83:
84:     if (pCurDefinitions == NULL)
85:         continue;
86:
87:     uiCurLength = strlen(pCurDefinitions);
88:
89:     if (uiCurSize + uiCurLength < uiBufferSize)
90:     {
91:         strcat (pReturnBuffer, pCurDefinitions);
92:         uiCurSize += uiCurLength;
93:     }
94:     else
95:         return;
96: }
97:
98: // Add variables from the second list, if they are already in the first list, continue
99: for (VariableDef* pCur = pDefs2; pCur != NULL; pCur = pCur->next)
100: {
101:     if (bIgnoreDuplicates && NULL != pDefs1)
102:     {
103:         if (pDefs1->ListContains(pCur->GetVariableName()))
104:             continue;
105:     }
106:
107:     const char* pCurDefinitions = pCur->GetDefinitions();
108:
109:     if (pCurDefinitions == NULL)
110:         continue;
111:
112:     uiCurLength = strlen(pCurDefinitions);
113:
114:     if (uiCurSize + uiCurLength < uiBufferSize)
115:     {
116:         strcat (pReturnBuffer, pCurDefinitions);
117:         uiCurSize += uiCurLength;
118:     }
119:     else

```

```

120:     return;
121: }
122: }
123:
124: void VariableDef::ToStringConcatVariableDefs(char* strReturnBuffer, unsigned int uiBufferSize,
125:                                             VariableDef* pDefs1, VariableDef* pDefs2)
126: {
127:     ToStringConcatVariableDefs(strReturnBuffer, uiBufferSize, pDefs1, pDefs2, false);
128: }
129:
130: void VariableDef::ToStringConcatVariableDefsIgnoreDuplicates(char* pReturnBuffer,
131:                                                             unsigned int uiBufferSize,
132:                                                             VariableDef* pDefs1,
133:                                                             VariableDef* pDefs2)
134: {
135:     ToStringConcatVariableDefs(pReturnBuffer, uiBufferSize, pDefs1, pDefs2, true);
136: }
137:
138: bool VariableDef::ListContains(const char* strVariableName) const
139: {
140:     if (0 == strcmp(this->GetVariableName(), strVariableName))
141:         return true;
142:
143:     for (VariableDef* pCur = next; pCur != NULL; pCur = pCur->next)
144:     {
145:         if (0 == strcmp(pCur->GetVariableName(), strVariableName))
146:         {
147:             return true;
148:         }
149:     }
150:
151:     return false;
152: }
153:
154: void VariableDef::Print()
155: {
156:     if (m_pDef == NULL)
157:         return;
158:
159:     printf("variable_def: %s interface_name: %s", m_pDef, m_strInterfaceName);
160:     if(next!=NULL)

```

```
161:     next->Print();  
162: }
```

File: sdm/common/xTEDS/VariableDef.h

```
1: #ifndef __SDM_VARIABLE_DEF_H
2: #define __SDM_VARIABLE_DEF_H
3: #include "../sdmLib.h"
4:
5: #include "../message_defs.h"
6:
7: class SDMLIB_API VariableDef
8: {
9: public:
10: VariableDef();
11: VariableDef(const VariableDef&);
12: ~VariableDef();
13:
14: VariableDef& operator=(const VariableDef&);
15:
16: void Join(VariableDef*);
17: const char* GetInterfaceName() const { return m_strInterfaceName; }
18: const char* GetVariableName() const { return m_strVariableName; }
19: const char* GetDefinitions() const { return m_pDef; }
20: void SetDefinitions(const char* strNewDefinitions);
21: void SetInterfaceName(const char* strNewInterfaceName);
22: void SetVariableName(const char* strNewVariableName);
23: bool ListContains(const char* strVariableName) const;
24:
25: static void ToStringConcatVariableDefs(char* strReturnBuffer,
26:                                     unsigned int uiReturnBufferSize,
27:                                     VariableDef* Defs1,
28:                                     VariableDef* Defs2);
29: static void ToStringConcatVariableDefsIgnoreDuplicates(char* strReturnBuffer,
30:                                     unsigned int uiBufferSize,
31:                                     VariableDef* Defs1,
32:                                     VariableDef* Defs2);
33:
34: void Print();
35: private:
36: static void ToStringConcatVariableDefs(char* strReturnBuffer, unsigned int uiReturnBufferSize,
37:                                     VariableDef* Defs1, VariableDef* Defs2, bool bIgnoreDulicates);
38: char* m_pDef;
39: char m_strInterfaceName[33];
```

```
40: char m_strVariableName[33];
41: public:
42: VariableDef* next;
43:
44: };
45:
46:
47: #endif
```

File: sdm/common/xTEDS/xTEDSCoefList.h

```
1: #ifndef __SDM_XTEDS_COEF_LIST_H_
2: #define __SDM_XTEDS_COEF_LIST_H_
3:
4: #include "xTEDSCoef.h"
5: extern "C"
6: {
7: #include "xTEDSParser.h"
8: }
9: #include <stdlib.h>
10: #include <cstring>
11:
12: struct xTEDSCoefListNode
13: {
14: xTEDSCoef data;
15: struct xTEDSCoefListNode* next;
16: xTEDSCoefListNode():data(),next(NULL) {}
17: xTEDSCoefListNode(const xTEDSCoefListNode&);
18: xTEDSCoefListNode& operator=(const xTEDSCoefListNode&);
19: };
20:
21: class xTEDSCoefList
22: {
23: public:
24: xTEDSCoefList();
25: xTEDSCoefList(const xTEDSCoefList&);
26: xTEDSCoefList& operator=(const xTEDSCoefList&);
27: ~xTEDSCoefList();
28:
29: void setCoefList(const coef* CoefList);
30: void addCoef(const coef* NewCoef);
31:
32: bool IsEmpty(void) const;
33: void VarInfoRequest(char* InfoBufferOut, size_t BufferSize) const;
34: private:
35: void deleteList();
36: struct xTEDSCoefListNode* head;
37: struct xTEDSCoefListNode* tail;
38: };
39:
```


40: #endif

File: sdm/common/xTEDS/xTEDSCoefList.cpp

```
1: #include "xTEDSCoefList.h"
2: #include "MessageDef.h"
3:
4: #include <stdlib.h>
5: #include <string.h>
6: #include <stdio.h>
7:
8: void xTEDSCoefList::deleteList()
9: {
10:  xTEDSCoefListNode *Temp;
11:  for (xTEDSCoefListNode *Cur = head; Cur != NULL; )
12:  {
13:      Temp = Cur->next;
14:      delete Cur;
15:      Cur = Temp;
16:  }
17: }
18:
19: xTEDSCoefList::xTEDSCoefList():head(NULL),tail(NULL)
20: {}
21:
22: xTEDSCoefList::~xTEDSCoefList()
23: {
24:  deleteList();
25: }
26:
27: void xTEDSCoefList::setCoefList(const coef* CoefList)
28: {
29:  if (CoefList == NULL) return;
30:
31:  addCoef(CoefList);
32:  for (coef* Cur = CoefList->next; Cur != NULL; Cur = Cur->next)
33:  {
34:      addCoef(Cur);
35:  }
36: }
37:
38: void xTEDSCoefList::addCoef(const coef* NewCoef)
39: {
```

```

40: if (NewCoef == NULL) return;
41:
42: xTEDSCoefListNode *NewNode = new xTEDSCoefListNode();
43: NewNode->data.setCoef(NewCoef);
44: if(head == NULL)
45: {
46:     head = NewNode;
47:     tail = NewNode;
48: }
49: else
50: {
51:     xTEDSCoefListNode* Cur;
52:     for (Cur = head; Cur->next != NULL; Cur = Cur->next)
53:         ;
54:     Cur->next = NewNode;
55:     tail = NewNode;
56: }
57: }
58:
59: bool xTEDSCoefList::IsEmpty() const
60: {
61:     return (head == NULL);
62: }
63:
64: void xTEDSCoefList::VarInfoRequest( char* InfoBufferOut, size_t BufferSize ) const
65: {
66:     const unsigned int MAX_BUF_SIZE = 1024;
67:     char Buf[MAX_BUF_SIZE];
68:
69:     if (head == NULL)
70:         return ;
71:
72:     Buf[0] = '\0';
73:     for (xTEDSCoefListNode* Cur = head; Cur != NULL; Cur = Cur->next)
74:     {
75:         strncat (Buf, " \n \t \t", sizeof(Buf) - strlen(Buf));
76:
77:         const size_t CurBufLength = strlen(Buf);
78:         Cur->data.VarInfoRequest(Buf + CurBufLength, sizeof(Buf) - CurBufLength);
79:     }
80:     strncat(InfoBufferOut, Buf, BufferSize - 1);

```

```

81: }
82:
83:
84: /* Not used..
85: xTEDSCoefList::xTEDSCoefList(const xTEDSCoefList& b):head(NULL),tail(NULL)
86: {
87: head = copyList(b.head,&tail);
88: }
89: xTEDSCoefList& xTEDSCoefList::operator=(const xTEDSCoefList& b)
90: {
91: deleteList(head);
92: head = copyList(b.head,&tail);
93: return *this;
94: }*/
95: /*struct xTEDSCoefListNode* copyList(struct xTEDSCoefListNode* list,struct
xTEDSCoefListNode** tail)
96: {
97: struct xTEDSCoefListNode* p;
98: struct xTEDSCoefListNode* head;
99: head = (struct xTEDSCoefListNode*)malloc(sizeof(struct xTEDSCoefListNode));
100: p = head;
101: for(struct xTEDSCoefListNode* cur=list;cur!=NULL;cur=cur->next)
102: {
103: p->data = cur->data;
104: if(cur->next!=NULL)
105: {
106: p->next = (struct xTEDSCoefListNode*)malloc(sizeof(struct xTEDSCoefListNode));
107: }
108: else
109: {
110: p->next = NULL;
111: *tail = p;
112: }
113: p = p->next;
114: }
115: return head;
116: }*/
117:

```

File: sdm/common/xTEDS/xTEDSItemTree.h

```
1: #ifndef __SDM_XTEDS_ITEM_TREE_H_
2: #define __SDM_XTEDS_ITEM_TREE_H_
3:
4:
5: #include "xTEDSItem.h"
6: #include "xTEDSQualifierList.h"
7: #include "xTEDSVariableList.h"
8: #include "xTEDSWrapper.h"
9: #include "xTEDSWrapperList.h"
10: #include "xTEDSVariable.h"
11: #include "xTEDSCommand.h"
12: #include "xTEDSRequest.h"
13: #include "xTEDSNotification.h"
14: #include "../sdmLib.h"
15:
16: class SDMLIB_API xTEDSItemTree
17: {
18: public:
19: xTEDSItemTree();
20: xTEDSItemTree(const xTEDSItemTree&);    //Not defined - shouldn't be used
21: xTEDSItemTree& operator=(const xTEDSItemTree&); //Not defined - shouldn't be used
22:
23: ~xTEDSItemTree();
24: MessageDef* ExactRegInfo(const char* Name, const xTEDSQualifierList& qualifier, const char*
Interface) const;
25: MessageDef* AllRegInfo(const xTEDSQualifierList& qualifier, const char* Interface) const;
26: MessageDef* RegexRegInfo(const char* Pattern, const xTEDSQualifierList& qualifier, const char*
Interface) const;
27:
28: void AddVariable(const variable* NewVariable);
29: bool AddCommand(const command* Command);
30: bool AddNotification(const notification* Notification);
31: bool AddRequest(const request* Request);
32:
33: bool IsCommandIdValid(const SDMMessage_ID& RequestedId) const { return
m_Commands.ContainsMessage(RequestedId); }
34: bool IsServiceIdValid(const SDMMessage_ID& RequestedId) const;
35:
36: SDMMessage_ID GetNotificationFaultID(const SDMMessage_ID& DataMessageID) const;
37: SDMMessage_ID GetCommandFaultID(const SDMMessage_ID& CommandMessageID) const;
```

```
38: SDMMMessage_ID GetServiceFaultID(const SDMMMessage_ID& CommandMessageID) const;
39: SDMMMessage_ID GetServiceDataID(const SDMMMessage_ID& CommandMessageID) const;
40: VariableDef* VarInfoRequest(const char* VarName, const SDMMMessage_ID& Interface) const;
41:
42: void PrintDebug();
43: private:
44: xTEDSVariableList m_Variables;
45: xTEDSWrapperList m_Commands;
46: xTEDSWrapperList m_Notifications;
47: xTEDSWrapperList m_Requests;
48: };
49:
50: #endif
```

File: sdm/common/xTEDS/xTEDS.y

```
1: %{
2: /*xTEDS 1.0 msg_def parser*/
3: #include <stdio.h>
4: #include <stdlib.h>
5: #include <string.h>
6:
7: #include "xTEDSParser.h"
8:
9: int yylex();
10:
11: int yydebug=0;
12: void yyerror(char *s);
13: %}
14:
15: /*%pure-parser*/
16: /*punctuation*/
17:
18: %token EQUAL_SY CLOSE_SY SLASHCLOSE_SY OPEN_XML_SY CLOSE_xTEDS_SY
OPEN_xTEDS_SY OPEN_APP_SY
19: %token OPEN_VAR_SY CLOSE_VAR_SY OPEN_DRANGE_SY CLOSE_DRANGE_SY
OPEN_OPTION_SY OPEN_CURVE_SY
20: %token CLOSE_CURVE_SY OPEN_COEFF_SY OPEN_DATA_MSG_SY
CLOSE_DATA_MSG_SY OPEN_VARIABLE_REF_SY
21: %token OPEN_COMMAND_MSG_SY CLOSE_COMMAND_MSG_SY NAME_SY KIND_SY
ID_SY CLOSE_ORIENTATION_SY
22: %token QUALIFIER_SY DESCRIPTION_SY MANUFACTURER_ID_SY VERSION_SY
MODEL_ID_SY VERSION_LETTER_SY
23: %token SERIAL_NUMBER_SY CALIBRATION_DATE_SY SENSITIVITY_AT_REF_SY
REF_FREQ_SY REF_TEMP_SY
24: %token MEASUREMENT_RANGE_SY ELECTRICAL_OUTPUT_SY QUALITY_FACTOR_SY
TEMP_COEFF_SY DIRECTION_XYZ_SY
25: %token CAL_DUE_DATE_SY POWER_REQS_SY VALUE_SY ALARM_SY
MSG_ARRIVAL_SY MSG_RATE_SY
26: %token STRING FLOAT INT PRECISION_SY RANGE_MAX_SY CLOSE_LOCATION_SY
CLOSE_ORIENTATION_SY
27: %token FORMAT_SY ACCURACY_SY RANGE_MIN_SY SCALE_FACTOR_SY UNITS_SY
DEFAULT_VALUE_SY
28: %token OPEN_DEVICE_SY SCALE_UNITS_SY LENGTH_SY EXPONENT_SY
SCHEMA_LOCATION_SY XMLNS_SY XMLNS_XSI_SY
29: %token CLOSE_OPTION_SY OPEN_INTERFACE_SY OPEN_COMMAND_SY
OPEN_NOTIFICATION_SY OPEN_REQUEST_SY
```

```

30: %token OPEN_FAULT_MSG_SY COMPONENT_KEY_SY SPA_U_HUB_SY SPA_U_PORT_SY
EXTENDS_SY
31: %token CLOSE_COMMAND_SY CLOSE_NOTIFICATION_SY CLOSE_REQUEST_SY
CLOSE_FAULT_MSG_SY OPEN_QUALIFIER_SY
32: %token CLOSE_QUALIFIER_SY CLOSE_APP_SY CLOSE_DEVICE_SY
CLOSE_INTERFACE_SY MEMORY_MINIMUM_SY
33: %token OPERATING_SYSTEM_SY PATH_FOR_ASSEMBLY_SY
PATH_ON_SPACECRAFT_SY X_SY Y_SY Z_SY AXIS_SY ANGLE_SY
34: %token OPEN_LOCATION_SY OPEN_ORIENTATION_SY CLOSE_XML_SY ENCODING_SY
STANDALONE_SY CLOSE_VARIABLE_REF_SY
35: %token CLOSE_COEFF_SY R_LOW_SY R_HIGH_SY Y_LOW_SY Y_HIGH_SY
INVALID_VALUE_SY BAD_TERMINAL_SY
36:
37: %union
38: {
39: int integer;
40: float real;
41: char* str;
42: struct variable_data* var;
43: struct variable_reference* var_ref;
44: struct qualifier_data* qual;
45: struct coefficient_data* coef;
46: struct curve_data* curve;
47: struct option_data* curveoption;
48: struct drange_data* drange;
49: struct location_data* location;
50: struct orientation_data* orientation;
51: struct fault_message* fault_msg;
52: struct data_message* data_msg;
53: struct command_message* cmd_msg;
54: struct command_type* command;
55: struct notification_type* notification;
56: struct request_type* request;
57: struct message_type* message;
58: struct interface_type* interface;
59: struct xteds* xteds;
60: struct app_device_attributes* attr;
61: }
62:
63: %type<str> STRING
64: %type<integer> INT ENCODING
65: %type<real> FLOAT

```


66: %type<var> VAR_ATTRIBUTES VAR_ATTRIBUTE VAR_SECTION VARIABLE VAR_HEAD
 VAR_WITH_SUBELEMENTS VAR_NO_SUBELEMENTS VAR_SUBELEMENT
 VAR_SUBELEMENTS
 67: %type<var> VAR_ELEMENTS VAR_QUALIFIERS
 68: %type<var_ref> VARIABLE_REFS VARIABLE_REF
 69: %type<qual> QUALIFIERS_SECTION QUALIFIER
 70: %type<qual> QUALIFIERS_ATTRIBUTE QUALIFIERS_ATTRIBUTES
 INTERFACE_SUBELEMENTS INTERFACE_SUBELEMENT
 71: %type<coef> CURVE_COEFFS CURVE_COEFF COEFF_ATTRIBUTES COEFF_ATTRIBUTE
 72: %type<curve> CURVE_HEAD CURVE_ATTRIBUTES CURVE_ATTRIBUTE CURVE
 73: %type<curveoption> DRANGE_OPTIONS DRANGE_OPTION OPTION_ATTRIBUTES
 OPTION_ATTRIBUTE
 74: %type<drange> DRANGE_HEAD DRANGE_ATTRIBUTES DRANGE_ATTRIBUTE DRANGE
 75: %type<location> LOCATION_SECTION LOCATION_ATTRIBUTES LOCATION_ATTRIBUTE
 76: %type<orientation> ORIENTATION_SECTION ORIENTATION_ATTRIBUTES
 ORIENTATION_ATTRIBUTE
 77: %type<fault_msg> FAULT_MSG_SECTION FAULT_MSG FAULT_MSG_ATTRIBUTES
 FAULT_MSG_ATTRIBUTE FAULT_MSG_QUALIFIERS
 78: %type<data_msg> DATA_MSG_SECTION DATA_MSG DATA_MSG_ATTRIBUTES
 DATA_MSG_ATTRIBUTE DATA_MSG_QUALIFIERS
 79: %type<cmd_msg> COMMAND_MSG_SECTION COMMAND_MSG
 COMMAND_MSG_ATTRIBUTES COMMAND_MSG_ATTRIBUTE
 COMMAND_MSG_QUALIFIERS
 80: %type<command> COMMAND_SECTION
 81: %type<notification> NOTIFICATION_SECTION
 82: %type<request> REQUEST_SECTION
 83: %type<message> MESSAGE_SECTION MESSAGES
 84: %type<interface> INTERFACE INTERFACES INTERFACE_HEAD INTERFACE_ATTRIBUTES
 INTERFACE_ATTRIBUTE
 85: %type<xteds> xTEDS OPEN_xTEDS xTEDS_ATTRIBUTES xTEDS_ATTRIBUTE
 86: %type<attr> APP_DEVICE APP_SECTION DEVICE_SECTION APP_ATTRIBUTES
 COMMON_APP_DEVICE_ATTRIBUTE
 87: %type<attr> DEVICE_ATTRIBUTES DEVICE_ATTRIBUTE APP_ATTRIBUTE
 DEVICE_SUBELEMENTS DEVICE_SUBELEMENT
 88:
 89: %start XTEDS_DOCUMENT
 90:
 91: %%
 92: XTEDS_DOCUMENT : OPEN_XML_SY VERSION_SY EQUAL_SY STRING
 ENCODING STANDALONE CLOSE_XML_SY xTEDS
 93: {
 94: int compare = 1;
 95: if(strcmp("1.0",\$4)!=0)
 96: compare = 0;

```

97:         free($4);
98:         compare = $5;
99:         if(compare == 1)
100:             result = $8;
101:         else
102:             result = NULL;
103:     }
104:     ;
105:
106: ENCODING      :   ENCODING_SY EQUAL_SY STRING
107:     {
108:         int compare = 1;
109:
110:         if(strcmp("utf-8",$3)!=0 && strcmp("UTF-8",$3)!=0)
111:             compare = 0;
112:         free($3);
113:         $$ = compare;
114:     }
115:     |   /*empty*/
116:     {
117:         $$ = 1;
118:     }
119:     ;
120:
121: STANDALONE    :   STANDALONE_SY EQUAL_SY STRING
122:     {
123:         free($3);
124:     }
125:     |   /*empty*/
126:     ;
127: //////////////////////////////////////
128: //
129: //      Productions pertaining to xTEDS structure in xTEDS
130: //
131: //////////////////////////////////////
132:
133: xTEDS         :   OPEN_xTEDS   APP_DEVICE   INTERFACES   CLOSE_xTEDS_SY
CLOSE_SY
134:     {
135:         $$ = add_references($1,$2,$3);
136:     }

```

```

137:      ;
138:
139: OPEN_xTEDS      :   OPEN_xTEDS_SY xTEDS_ATTRIBUTES CLOSE_SY
140:      {
141:          $$=$2;
142:      }
143:      ;
144:
145: xTEDS_ATTRIBUTES  :   xTEDS_ATTRIBUTES xTEDS_ATTRIBUTE
146:      {
147:          $$ = merge_xteds($1,$2);
148:      }
149:      | /*empty*/
150:      {
151:          xteds* temp;
152:          temp = new_xteds();
153:          $$=temp;
154:      }
155:      ;
156:
157: xTEDS_ATTRIBUTE   :   NAME_SY EQUAL_SY STRING
158:      {
159:          xteds* temp;
160:          temp = new_xteds();
161:          temp->name = $3;
162:          $$ = temp;
163:      }
164:      | VERSION_SY EQUAL_SY STRING
165:      {
166:          xteds* temp;
167:          temp = new_xteds();
168:          temp->version = $3;
169:          $$ = temp;
170:      }
171:      | DESCRIPTION_SY EQUAL_SY STRING
172:      {
173:          xteds* temp;
174:          temp = new_xteds();
175:          temp->description = $3;
176:          $$ = temp;
177:      }

```

```

178:      | XMLNS_SY EQUAL_SY STRING
179:      {
180:          xteds* temp;
181:          temp = new_xteds();
182:          temp->xmlns = $3;
183:          $$ = temp;
184:      }
185:      | SCHEMA_LOCATION_SY EQUAL_SY STRING
186:      {
187:          xteds* temp;
188:          temp = new_xteds();
189:          temp->schema_location = $3;
190:          $$ = temp;
191:      }
192:      | XMLNS_XSI_SY EQUAL_SY STRING
193:      {
194:          xteds* temp;
195:          temp = new_xteds();
196:          temp->xmlns_xsi = $3;
197:          $$ = temp;
198:      }
199:      ;
200: //////////////////////////////////////
201: //
202: //      Productions pertaining to application definition in xTEDS
203: //
204: //////////////////////////////////////
205:
206: APP_DEVICE      :   APP_SECTION
207:      {
208:          $$ = $1;
209:      }
210:      |   DEVICE_SECTION
211:      {
212:          $$ = $1;
213:      }
214:      ;
215:
216: APP_SECTION      :   OPEN_APP_SY APP_ATTRIBUTES SLASHCLOSE_SY
217:      {
218:          $$ = $2;

```

```

219:         }
220:         | OPEN_APP_SY APP_ATTRIBUTES CLOSE_SY QUALIFIERS_SECTION
CLOSE_APP_SY CLOSE_SY
221:         {
222:             app_dev_attr* temp;
223:             temp = new_app_dev_attr();
224:             temp->qualifiers = $4;
225:             $$ = merge_app_dev_attr($2,temp);
226:         }
227:         ;
228:
229: APP_ATTRIBUTES      : APP_ATTRIBUTES APP_ATTRIBUTE
230:         {
231:             $$ = merge_app_dev_attr($1,$2);
232:         }
233:         | /*empty*/
234:         {
235:             $$ = NULL;
236:         }
237:         ;
238:
239: COMMON_APP_DEVICE_ATTRIBUTE: NAME_SY EQUAL_SY STRING
240:         {
241:             app_dev_attr* temp;
242:             temp = new_app_dev_attr();
243:             temp->name = $3;
244:             $$ = temp;
245:         }
246:         | KIND_SY EQUAL_SY STRING
247:         {
248:             app_dev_attr* temp;
249:             temp = new_app_dev_attr();
250:             temp->kind = $3;
251:             $$ = temp;
252:         }
253:         | ID_SY EQUAL_SY STRING
254:         {
255:             app_dev_attr* temp;
256:             temp = new_app_dev_attr();
257:             temp->id = $3;
258:             $$ = temp;

```

```

259:     }
260:     |   QUALIFIER_SY EQUAL_SY STRING
261:     {
262:         app_dev_attr* temp;
263:         temp = new_app_dev_attr();
264:         temp->qualifier = $3;
265:         printf("Qualifier field has been deprecated! \n");
266:         $$ = temp;
267:     }
268:     |   DESCRIPTION_SY EQUAL_SY STRING
269:     {
270:         app_dev_attr* temp;
271:         temp = new_app_dev_attr();
272:         temp->description = $3;
273:         $$ = temp;
274:     }
275:     |   MANUFACTURER_ID_SY EQUAL_SY STRING
276:     {
277:         app_dev_attr* temp;
278:         temp = new_app_dev_attr();
279:         temp->manufacturer_id = $3;
280:         $$ = temp;
281:     }
282:     |   COMPONENT_KEY_SY EQUAL_SY STRING
283:     {
284:         app_dev_attr* temp;
285:         temp = new_app_dev_attr();
286:         temp->componentKey = $3;
287:         $$ = temp;
288:     }
289:     ;
290:
291: APP_ATTRIBUTE      :   COMMON_APP_DEVICE_ATTRIBUTE
292:     {
293:         $$ = $1;
294:     }
295:     |   VERSION_SY EQUAL_SY STRING
296:     {
297:         app_dev_attr* temp;
298:         temp = new_app_dev_attr();
299:         temp->version = $3;

```

```

300:         $$ = temp;
301:     }
302:     | MEMORY_MINIMUM_SY EQUAL_SY STRING
303:     {
304:         app_dev_attr* temp;
305:         temp = new_app_dev_attr();
306:         temp->memory_minimum = $3;
307:         $$ = temp;
308:     }
309:     | OPERATING_SYSTEM_SY EQUAL_SY STRING
310:     {
311:         app_dev_attr* temp;
312:         temp = new_app_dev_attr();
313:         temp->operating_system = $3;
314:         $$ = temp;
315:     }
316:     | PATH_FOR_ASSEMBLY_SY EQUAL_SY STRING
317:     {
318:         app_dev_attr* temp;
319:         temp = new_app_dev_attr();
320:         temp->path_for_assembly = $3;
321:         $$ = temp;
322:     }
323:     | PATH_ON_SPACECRAFT_SY EQUAL_SY STRING
324:     {
325:         app_dev_attr* temp;
326:         temp = new_app_dev_attr();
327:         temp->path_on_spacecraft = $3;
328:         $$ = temp;
329:     }
330:     ;
331: //////////////////////////////////////
332: //
333: //      Productions pertaining to device definition in xTEDS
334: //
335: //////////////////////////////////////
336:
337: DEVICE_SECTION      :   OPEN_DEVICE_SY DEVICE_ATTRIBUTES SLASHCLOSE_SY
338:     {
339:         $$ = $2;
340:     }

```

```

341:      | OPEN_DEVICE_SY      DEVICE_ATTRIBUTES      CLOSE_SY
DEVICE_SUBELEMENTS CLOSE_DEVICE_SY CLOSE_SY QUALIFIERS_SECTION
342:      {
343:          app_dev_attr* temp;
344:          temp = new_app_dev_attr();
345:          temp->qualifiers = $7;
346:          merge_app_dev_attr($2,temp);
347:          $$ = merge_app_dev_attr($2,$4);
348:      }
349:      ;
350:
351: DEVICE_SUBELEMENTS : DEVICE_SUBELEMENTS DEVICE_SUBELEMENT
352:      {
353:          $$ = merge_app_dev_attr($1,$2);
354:      }
355:      | /*empty*/
356:      {
357:          $$ = NULL;
358:      }
359:      ;
360:
361: DEVICE_SUBELEMENT : QUALIFIER
362:      {
363:          app_dev_attr* temp = new_app_dev_attr();
364:          temp->qualifiers = $1;
365:          $$ = temp;
366:      }
367:      | LOCATION_SECTION
368:      {
369:          delete_location($1); //Location not used
370:          $$ = NULL;
371:      }
372:      | ORIENTATION_SECTION
373:      {
374:          delete_orientation($1); //Orientation not used
375:          $$ = NULL;
376:      }
377:      ;
378:
379: DEVICE_ATTRIBUTES : DEVICE_ATTRIBUTES DEVICE_ATTRIBUTE
380:      {

```



```

381:         $$ = merge_app_dev_attr($1,$2);
382:     }
383:     | /*empty*/
384:     {
385:         $$ = NULL;
386:     }
387:     ;
388:
389: DEVICE_ATTRIBUTE    :    COMMON_APP_DEVICE_ATTRIBUTE
390:     {
391:         $$ = $1;
392:     }
393:     | MODEL_ID_SY EQUAL_SY STRING
394:     {
395:         app_dev_attr* temp;
396:         temp = new_app_dev_attr();
397:         temp->model_id = $3;
398:         $$ = temp;
399:     }
400:     | VERSION_LETTER_SY EQUAL_SY STRING
401:     {
402:         app_dev_attr* temp;
403:         temp = new_app_dev_attr();
404:         temp->version_letter = $3;
405:         $$ = temp;
406:     }
407:     | SERIAL_NUMBER_SY EQUAL_SY STRING
408:     {
409:         app_dev_attr* temp;
410:         temp = new_app_dev_attr();
411:         temp->serial_number = $3;
412:         $$ = temp;
413:     }
414:     | CALIBRATION_DATE_SY EQUAL_SY STRING
415:     {
416:         app_dev_attr* temp;
417:         temp = new_app_dev_attr();
418:         temp->calibration_date = $3;
419:         $$ = temp;
420:     }
421:     | SENSITIVITY_AT_REF_SY EQUAL_SY STRING

```

```

422:      {
423:          app_dev_attr* temp;
424:          temp = new_app_dev_attr();
425:          temp->sensitivity_at_reference = $3;
426:          $$ = temp;
427:      }
428:      | REF_FREQ_SY EQUAL_SY STRING
429:      {
430:          app_dev_attr* temp;
431:          temp = new_app_dev_attr();
432:          temp->reference_frequency = $3;
433:          $$ = temp;
434:      }
435:      | REF_TEMP_SY EQUAL_SY STRING
436:      {
437:          app_dev_attr* temp;
438:          temp = new_app_dev_attr();
439:          temp->reference_temperature = $3;
440:          $$ = temp;
441:      }
442:      | MEASUREMENT_RANGE_SY EQUAL_SY STRING
443:      {
444:          app_dev_attr* temp;
445:          temp = new_app_dev_attr();
446:          temp->measurement_range = $3;
447:          $$ = temp;
448:      }
449:      | ELECTRICAL_OUTPUT_SY EQUAL_SY STRING
450:      {
451:          app_dev_attr* temp;
452:          temp = new_app_dev_attr();
453:          temp->electrical_output = $3;
454:          $$ = temp;
455:      }
456:      | QUALITY_FACTOR_SY EQUAL_SY STRING
457:      {
458:          app_dev_attr* temp;
459:          temp = new_app_dev_attr();
460:          temp->quality_factor = $3;
461:          $$ = temp;
462:      }

```

```

463: | TEMP_COEFF_SY EQUAL_SY STRING
464: {
465:     app_dev_attr* temp;
466:     temp = new_app_dev_attr();
467:     temp->temperature_coefficient = $3;
468:     $$ = temp;
469: }
470: | DIRECTION_XYZ_SY EQUAL_SY STRING
471: {
472:     app_dev_attr* temp;
473:     temp = new_app_dev_attr();
474:     temp->direction_xyz = $3;
475:     $$ = temp;
476: }
477: | CAL_DUE_DATE_SY EQUAL_SY STRING
478: {
479:     app_dev_attr* temp;
480:     temp = new_app_dev_attr();
481:     temp->cal_due_date = $3;
482:     $$ = temp;
483: }
484: | POWER_REQS_SY EQUAL_SY STRING
485: {
486:     app_dev_attr* temp;
487:     temp = new_app_dev_attr();
488:     temp->power_requirements = $3;
489:     $$ = temp;
490: }
491: | SPA_U_HUB_SY EQUAL_SY STRING
492: {
493:     app_dev_attr* temp;
494:     temp = new_app_dev_attr();
495:     temp->spa_u_hub = $3;
496:     $$ = temp;
497: }
498: | SPA_U_PORT_SY EQUAL_SY STRING
499: {
500:     app_dev_attr* temp;
501:     temp = new_app_dev_attr();
502:     temp->spa_u_port = $3;
503:     $$ = temp;

```

```

504:      }
505:      |   VERSION_SY EQUAL_SY STRING
506:      {
507:          app_dev_attr* temp;
508:          temp = new_app_dev_attr();
509:          temp->version = $3;
510:          $$ = temp;
511:      }
512:      ;
513: //////////////////////////////////////
514: //
515: //      Productions pertaining to location definition in xTEDS
516: //
517: //////////////////////////////////////
518:
519: LOCATION_SECTION  :   OPEN_LOCATION_SY              LOCATION_ATTRIBUTES
SLASHCLOSE_SY
520:      {
521:          $$ = $2;
522:      }
523:      |   OPEN_LOCATION_SY              LOCATION_ATTRIBUTES              CLOSE_SY
CLOSE_LOCATION_SY CLOSE_SY
524:      {
525:          $$ = $2;
526:      }
527:      ;
528:
529: LOCATION_ATTRIBUTES :   LOCATION_ATTRIBUTES LOCATION_ATTRIBUTE
530:      {
531:          $$ = merge_locations($1,$2);
532:      }
533:      |   /*empty*/
534:      {
535:          $$ = NULL;
536:      }
537:      ;
538:
539: LOCATION_ATTRIBUTE  :   X_SY EQUAL_SY STRING
540:      {
541:          location* x_loc = new_location();
542:          x_loc->x_value = $3;

```

```

543:         $$ = x_loc;
544:     }
545:     |   Y_SY EQUAL_SY STRING
546:     {
547:         location* y_loc = new_location();
548:         y_loc->y_value = $3;
549:         $$ = y_loc;
550:     }
551:     |   Z_SY EQUAL_SY STRING
552:     {
553:         location* z_loc = new_location();
554:         z_loc->z_value = $3;
555:         $$ = z_loc;
556:     }
557:     |   UNITS_SY EQUAL_SY STRING
558:     {
559:         location* units_val = new_location();
560:         units_val->units = $3;
561:         $$ = units_val;
562:     }
563:     ;
564: //////////////////////////////////////
565: //
566: //      Productions pertaining to orientation definition in xTEDS
567: //
568: //////////////////////////////////////
569:
570: ORIENTATION_SECTION    :   OPEN_ORIENTATION_SY   ORIENTATION_ATTRIBUTES
SLASHCLOSE_SY
571:     {
572:         $$ = $2;
573:     }
574:     |   OPEN_ORIENTATION_SY      ORIENTATION_ATTRIBUTES      CLOSE_SY
CLOSE_ORIENTATION_SY CLOSE_SY
575:     {
576:         $$ = $2;
577:     }
578:     ;
579:
580: ORIENTATION_ATTRIBUTES :   ORIENTATION_ATTRIBUTES
ORIENTATION_ATTRIBUTE
581:     {

```

```

582:         $$ = merge_orientations($1,$2);
583:     }
584:     | /*empty*/
585:     {
586:         $$ = NULL;
587:     }
588:     ;
589:
590: ORIENTATION_ATTRIBUTE    :   AXIS_SY EQUAL_SY STRING
591:     {
592:         orientation* temp = new_orientation();
593:         temp->axis_value = $3;
594:         $$ = temp;
595:     }
596:     |   ANGLE_SY EQUAL_SY STRING
597:     {
598:         orientation* temp = new_orientation();
599:         temp->angle_value = $3;
600:         $$ = temp;
601:     }
602:     |   UNITS_SY EQUAL_SY STRING
603:     {
604:         orientation* temp = new_orientation();
605:         temp->units_value = $3;
606:         $$ = temp;
607:     }
608: //////////////////////////////////////
609: //
610: //      Productions pertaining to qualifier definition in xTEDS
611: //
612: //////////////////////////////////////
613:     ;
614: QUALIFIERS_SECTION :   QUALIFIERS_SECTION QUALIFIER
615:     {
616:         $$= link_qualifiers($1,$2);
617:     }
618:     | /*empty*/
619:     {
620:         $$= NULL;
621:     }
622:     ;

```

```

623:
624: QUALIFIER      :  OPEN_QUALIFIER_SY  QUALIFIERS_ATTRIBUTES  CLOSE_SY
CLOSE_QUALIFIER_SY CLOSE_SY
625:      {
626:          $$=$2
627:      }
628:      |  OPEN_QUALIFIER_SY QUALIFIERS_ATTRIBUTES SLASHCLOSE_SY
629:      {
630:          $$=$2
631:      }
632:      ;
633:
634: QUALIFIERS_ATTRIBUTES :  QUALIFIERS_ATTRIBUTES QUALIFIERS_ATTRIBUTE
635:      {
636:          $$ = merge_qualifiers($1,$2);
637:      }
638:      |  /*empty*/
639:      {
640:          $$ = NULL;
641:      }
642:      ;
643:
644: QUALIFIERS_ATTRIBUTE :  NAME_SY EQUAL_SY STRING
645:      {
646:          qualifier_type* temp;
647:          temp = new_qualifier();
648:          temp->name = $3;
649:          $$ = temp;
650:      }
651:      |  VALUE_SY EQUAL_SY STRING
652:      {
653:          qualifier_type* temp;
654:          temp = new_qualifier();
655:          temp->value = $3;
656:          $$ = temp;
657:      }
658:      |  UNITS_SY EQUAL_SY STRING
659:      {
660:          qualifier_type* temp;
661:          temp = new_qualifier();
662:          temp->units = $3;

```

```

663:          $$ = temp;
664:      }
665:      ;
666: //////////////////////////////////////
667: //
668: //      Productions pertaining to interface definition in xTEDS
669: //
670: //////////////////////////////////////
671:
672: INTERFACES      :   INTERFACES INTERFACE
673:      {
674:          $$ = link_interface($1,$2);
675:      }
676:      |   /*empty*/
677:      {
678:          $$ = NULL;
679:      }
680:      ;
681:
682: INTERFACE      :   INTERFACE_HEAD  INTERFACE_SUBELEMENTS  VAR_SECTION
MESSAGE_SECTION CLOSE_INTERFACE_SY CLOSE_SY
683:      {
684:          $$ = interface_add_references($1,$2,$3,$4);
685:      }
686:      |   INTERFACE_HEAD
687:      {
688:          $$ = $1;
689:      }
690:      ;
691:
692: INTERFACE_HEAD      :   OPEN_INTERFACE_SY              INTERFACE_ATTRIBUTES
CLOSE_SY
693:      {
694:          $$ = $2;
695:      }
696:      |   OPEN_INTERFACE_SY INTERFACE_ATTRIBUTES SLASHCLOSE_SY
697:      {
698:          $$ = $2;
699:      }
700:      ;
701:

```



```

702: INTERFACE_ATTRIBUTES :   INTERFACE_ATTRIBUTES INTERFACE_ATTRIBUTE
703:         {
704:             $$ = merge_interface($1,$2);
705:         }
706:         | /*empty*/
707:         {
708:             interface* temp;
709:             temp = new_interface();
710:             $$ = temp;
711:         }
712:         ;
713:
714: INTERFACE_SUBELEMENTS   :   INTERFACE_SUBELEMENTS
INTERFACE_SUBELEMENT
715:         {
716:             //Only qualifiers will be returned, all others thrown away
717:             $$ = link_qualifiers($1,$2);
718:         }
719:         | /*empty*/
720:         {
721:             $$ = NULL;
722:         }
723:         ;
724:
725: INTERFACE_SUBELEMENT:   QUALIFIER
726:         {
727:             $$ = $1;
728:         }
729:         | LOCATION_SECTION
730:         {
731:             // Don't need to save location information for an interface
732:             delete_location($1);
733:             $$ = NULL;
734:         }
735:         | ORIENTATION_SECTION
736:         {
737:             // Don't need to save orientation information for an interface
738:             delete_orientation($1);
739:             $$ = NULL;
740:         }
741:         ;

```

```

742:
743: INTERFACE_ATTRIBUTE : NAME_SY EQUAL_SY STRING
744:     {
745:         interface* temp;
746:         temp = new_interface();
747:         temp->name = $3;
748:         $$ = temp;
749:     }
750: | EXTENDS_SY EQUAL_SY STRING
751:     {
752:         interface* temp;
753:         temp = new_interface();
754:         temp->extends = $3;
755:         $$ = temp;
756:     }
757: | ID_SY EQUAL_SY STRING
758:     {
759:         interface* temp;
760:         temp = new_interface();
761:         temp->id = $3;
762:         $$ = temp;
763:     }
764: | DESCRIPTION_SY EQUAL_SY STRING
765:     {
766:         interface* temp;
767:         temp = new_interface();
768:         temp->description = $3;
769:         $$ = temp;
770:     }
771: ;
772: //////////////////////////////////////
773: //
774: //      Productions pertaining to message definition in xTEDS
775: //
776: //////////////////////////////////////
777:
778: MESSAGE_SECTION : MESSAGE_SECTION MESSAGES
779:     {
780:         $$ = merge_message($1,$2);
781:     }
782: | /*empty*/

```

```

783:      {
784:          $$ = NULL;
785:      }
786:      ;
787:
788: MESSAGES      :   COMMAND_SECTION
789:      {
790:          message* temp;
791:          temp = new_message();
792:          temp->commands = $1;
793:          $$ = temp;
794:      }
795:      |   NOTIFICATION_SECTION
796:      {
797:          message* temp;
798:          temp = new_message();
799:          temp->notifications = $1;
800:          $$ = temp;
801:      }
802:      |   REQUEST_SECTION
803:      {
804:          message* temp;
805:          temp = new_message();
806:          temp->requests = $1;
807:          $$ = temp;
808:      }
809:      ;
810:
811: REQUEST_SECTION      :   OPEN_REQUEST_SY CLOSE_SY COMMAND_MSG_SECTION
DATA_MSG_SECTION FAULT_MSG_SECTION CLOSE_REQUEST_SY CLOSE_SY
812:      {
813:          request* temp;
814:          temp = new_request();
815:          temp = request_add_cmd_msg(temp,$3);
816:          temp = request_add_data_msg(temp,$4);
817:          $$ = request_add_fault_msg(temp,$5);
818:      }
819:      ;
820: //////////////////////////////////////
821: //
822: //      Productions pertaining to fault messages in xTEDS

```

```

823: //
824: //////////////////////////////////////
825:
826: FAULT_MSG_SECTION :   FAULT_MSG
827:     {
828:         $$ = $1;
829:     }
830:     ;
831:
832: FAULT_MSG      :   OPEN_FAULT_MSG_SY   FAULT_MSG_ATTRIBUTES   CLOSE_SY
FAULT_MSG_QUALIFIERS VARIABLE_REFS CLOSE_FAULT_MSG_SY CLOSE_SY
833:     {
834:         fault_msg* result;
835:         result = merge_fault_msg($2,$4);
836:         $$ = fault_add_var_refs(result,$5);
837:     }
838:     |   OPEN_FAULT_MSG_SY FAULT_MSG_ATTRIBUTES SLASHCLOSE_SY
839:     {
840:         $$ = $2;
841:     }
842:     |   /*empty*/
843:     {
844:         $$ = NULL;
845:     }
846:     ;
847:
848: FAULT_MSG_ATTRIBUTES :   FAULT_MSG_ATTRIBUTES FAULT_MSG_ATTRIBUTE
849:     {
850:         $$ = merge_fault_msg($1,$2);
851:     }
852:     |   /*empty*/
853:     {
854:         $$ = NULL;
855:     }
856:     ;
857:
858: FAULT_MSG_ATTRIBUTE :   NAME_SY EQUAL_SY STRING
859:     {
860:         fault_msg* temp;
861:         temp = new_fault_msg();
862:         temp->name = $3;

```

```

863:         $$ = temp;
864:     }
865:     | ID_SY EQUAL_SY STRING
866:     {
867:         fault_msg* temp;
868:         temp = new_fault_msg();
869:         temp->id = $3;
870:         $$ = temp;
871:     }
872:     | DESCRIPTION_SY EQUAL_SY STRING
873:     {
874:         fault_msg* temp;
875:         temp = new_fault_msg();
876:         temp->description = $3;
877:         $$ = temp;
878:     }
879:     ;
880:
881: FAULT_MSG_QUALIFIERS :   QUALIFIERS_SECTION
882:     {
883:         fault_msg* temp;
884:         temp = new_fault_msg();
885:         temp->qualifiers = $1;
886:         $$ = temp;
887:     }
888:     ;
889: //////////////////////////////////////
890: //
891: //   Productions pertaining to data and notification messages in xTEDS
892: //
893: //////////////////////////////////////
894:
895: NOTIFICATION_SECTION :   OPEN_NOTIFICATION_SY                               CLOSE_SY
DATA_MSG_SECTION FAULT_MSG_SECTION CLOSE_NOTIFICATION_SY CLOSE_SY
896:     {
897:         notification* temp;
898:         temp = new_notification();
899:         temp = notification_add_data_msg(temp,$3);
900:         $$ = notification_add_fault_msg(temp,$4);
901:     }
902:     ;

```

```

903:
904: DATA_MSG_SECTION : DATA_MSG_SECTION DATA_MSG
905:     {
906:         $$ = link_data_msg($1,$2);
907:     }
908:     | /*empty*/
909:     {
910:         $$ = NULL;
911:     }
912:     ;
913:
914: DATA_MSG : OPEN_DATA_MSG_SY DATA_MSG_ATTRIBUTES CLOSE_SY
DATA_MSG_QUALIFIERS VARIABLE_REFS CLOSE_DATA_MSG_SY CLOSE_SY
915:     {
916:         data_msg* result;
917:         result = merge_data_msg($2,$4);
918:         $$ = data_add_var_refs(result,$5);
919:     }
920:     ;
921:
922: DATA_MSG_ATTRIBUTES : DATA_MSG_ATTRIBUTES DATA_MSG_ATTRIBUTE
923:     {
924:         $$=merge_data_msg($1,$2);
925:     }
926:     | /*empty*/
927:     {
928:         $$=NULL;
929:     }
930:     ;
931:
932: DATA_MSG_ATTRIBUTE : NAME_SY EQUAL_SY STRING
933:     {
934:         data_msg* temp;
935:         temp = new_data_msg();
936:         temp->name = $3;
937:         $$ = temp;
938:     }
939:     | ID_SY EQUAL_SY STRING
940:     {
941:         data_msg* temp;
942:         temp = new_data_msg();

```

```

943:         temp->id = $3;
944:         $$ = temp;
945:     }
946:     | MSG_ARRIVAL_SY EQUAL_SY STRING
947:     {
948:         data_msg* temp;
949:         temp = new_data_msg();
950:         temp->msg_arrival = $3;
951:         $$ = temp;
952:     }
953:     | DESCRIPTION_SY EQUAL_SY STRING
954:     {
955:         data_msg* temp;
956:         temp = new_data_msg();
957:         temp->description = $3;
958:         $$ = temp;
959:     }
960:     | MSG_RATE_SY EQUAL_SY STRING
961:     {
962:         data_msg* temp;
963:         temp = new_data_msg();
964:         temp->msg_rate = $3;
965:         $$ = temp;
966:     }
967:     ;
968:
969: DATA_MSG_QUALIFIERS : QUALIFIERS_SECTION
970:     {
971:         data_msg* temp;
972:         temp = new_data_msg();
973:         temp->qualifiers = $1;
974:         $$ = temp;
975:     }
976:     ;
977: //////////////////////////////////////
978: //
979: //      Productions pertaining to command messages in xTEDS
980: //
981: //////////////////////////////////////
982:

```

```

983: COMMAND_SECTION      :  OPEN_COMMAND_SY          CLOSE_SY
COMMAND_MSG_SECTION FAULT_MSG_SECTION CLOSE_COMMAND_SY CLOSE_SY
984:      {
985:          command* temp;
986:          temp = new_command();
987:          temp = command_add_cmd_msg(temp,$3);
988:          $$ = command_add_fault_msg(temp,$4);
989:      }
990:      ;
991:
992: COMMAND_MSG_SECTION    :  COMMAND_MSG_SECTION COMMAND_MSG
993:      {
994:          $$ = link_cmd_msg($1,$2);
995:      }
996:      | /*empty*/
997:      {
998:          $$ = NULL;
999:      }
1000:      ;
1001:
1002: COMMAND_MSG      :  OPEN_COMMAND_MSG_SY COMMAND_MSG_ATTRIBUTES
CLOSE_SY COMMAND_MSG_QUALIFIERS VARIABLE_REFS CLOSE_COMMAND_MSG_SY
CLOSE_SY
1003:      {
1004:          cmd_msg* result;
1005:          result = merge_cmd_msg($2,$4);
1006:          $$ = cmd_add_var_refs(result,$5);
1007:      }
1008:      | OPEN_COMMAND_MSG_SY          COMMAND_MSG_ATTRIBUTES
SLASHCLOSE_SY
1009:      {
1010:          $$ = cmd_add_var_refs($2,NULL);
1011:      }
1012:      ;
1013:
1014: COMMAND_MSG_ATTRIBUTES :  COMMAND_MSG_ATTRIBUTES
COMMAND_MSG_ATTRIBUTE
1015:      {
1016:          $$ = merge_cmd_msg($1,$2);
1017:      }
1018:      | /*empty*/
1019:      {

```



```

1020:         $$=NULL;
1021:     }
1022:     ;
1023:
1024: COMMAND_MSG_ATTRIBUTE    :    NAME_SY EQUAL_SY STRING
1025:     {
1026:         cmd_msg* temp;
1027:         temp = new_cmd_msg();
1028:         temp->name = $3;
1029:         $$ = temp;
1030:     }
1031: |    ID_SY EQUAL_SY STRING
1032:     {
1033:         cmd_msg* temp;
1034:         temp = new_cmd_msg();
1035:         temp->id = $3;
1036:         $$ = temp;
1037:     }
1038: |    DESCRIPTION_SY EQUAL_SY STRING
1039:     {
1040:         cmd_msg* temp;
1041:         temp = new_cmd_msg();
1042:         temp->description = $3;
1043:         $$ = temp;
1044:     }
1045:     ;
1046:
1047: COMMAND_MSG_QUALIFIERS    :    QUALIFIERS_SECTION
1048:     {
1049:         cmd_msg* temp;
1050:         temp = new_cmd_msg();
1051:         temp->qualifiers = $1;
1052:         $$ = temp;
1053:     }
1054:     ;
1055: //////////////////////////////////////
1056: //
1057: //      Productions pertaining to variable definition in xTEDS
1058: //
1059: //////////////////////////////////////
1060:

```

```

1061: VAR_SECTION      :   VAR_SECTION VARIABLE
1062:      {
1063:          $$= link_variables($1,$2);
1064:      }
1065:      |   /*empty*/
1066:      {
1067:          $$= NULL;
1068:      }
1069:      ;
1070:
1071: VARIABLE            :   VAR_WITH_SUBELEMENTS
1072:      {
1073:          $$=$1
1074:      }
1075:      |   VAR_NO_SUBELEMENTS
1076:      {
1077:          $$=$1
1078:      }
1079:      ;
1080:
1081: VAR_WITH_SUBELEMENTS :   VAR_HEAD          CLOSE_SY          VAR_ELEMENTS
CLOSE_VAR_SY CLOSE_SY
1082:      {
1083:          $$ = merge_variables($1,$3);
1084:      }
1085:      ;
1086:
1087: VAR_NO_SUBELEMENTS :   VAR_HEAD SLASHCLOSE_SY
1088:      ;
1089:
1090: VAR_HEAD            :   OPEN_VAR_SY VAR_ATTRIBUTES
1091:      {
1092:          $$= $2
1093:      }
1094:      ;
1095:
1096: VAR_ATTRIBUTES      :   VAR_ATTRIBUTES VAR_ATTRIBUTE
1097:      {
1098:          $$= merge_variables($1,$2);
1099:      }
1100:      |   /*empty*/

```

```

1101:      {
1102:          $$=NULL;
1103:      }
1104:      ;
1105:
1106: VAR_ATTRIBUTE      :   NAME_SY EQUAL_SY STRING
1107:      {
1108:          variable* temp;
1109:          temp = new_variable();
1110:          temp->name = $3;
1111:          $$ = temp;
1112:      }
1113:      |   KIND_SY EQUAL_SY STRING
1114:      {
1115:          variable* temp;
1116:          temp = new_variable();
1117:          temp->kind = $3;
1118:          $$ = temp;
1119:      }
1120:      |   FORMAT_SY EQUAL_SY STRING
1121:      {
1122:          variable* temp;
1123:          temp = new_variable();
1124:          temp->format = $3;
1125:          $$ = temp;
1126:      }
1127:      |   QUALIFIER_SY EQUAL_SY STRING
1128:      {
1129:          variable* temp;
1130:          temp = new_variable();
1131:          temp->qualifier = $3;
1132:          printf("<Variable qualifier attribute has been deprecated! \n");
1133:          $$ = temp;
1134:      }
1135:      |   ID_SY EQUAL_SY STRING
1136:      {
1137:          variable* temp;
1138:          temp = new_variable();
1139:          temp->id = $3;
1140:          $$ = temp;
1141:      }

```

```

1142:      | DESCRIPTION_SY EQUAL_SY STRING
1143:      {
1144:          variable* temp;
1145:          temp = new_variable();
1146:          temp->description = $3;
1147:          $$ = temp;
1148:      }
1149:      | RANGE_MIN_SY EQUAL_SY STRING
1150:      {
1151:          variable* temp;
1152:          temp = new_variable();
1153:          temp->range_min = $3;
1154:          $$ = temp;
1155:      }
1156:      | RANGE_MAX_SY EQUAL_SY STRING
1157:      {
1158:          variable* temp;
1159:          temp = new_variable();
1160:          temp->range_max = $3;
1161:          $$ = temp;
1162:      }
1163:      | LENGTH_SY EQUAL_SY STRING
1164:      {
1165:          variable* temp;
1166:          temp = new_variable();
1167:          temp->length = $3;
1168:          $$ = temp;
1169:      }
1170:      | DEFAULT_VALUE_SY EQUAL_SY STRING
1171:      {
1172:          variable* temp;
1173:          temp = new_variable();
1174:          temp->default_value = $3;
1175:          $$ = temp;
1176:      }
1177:      | PRECISION_SY EQUAL_SY STRING
1178:      {
1179:          variable* temp;
1180:          temp = new_variable();
1181:          temp->precision = $3;
1182:          $$ = temp;

```

```

1183:     }
1184:     |   UNITS_SY EQUAL_SY STRING
1185:     {
1186:         variable* temp;
1187:         temp = new_variable();
1188:         temp->units = $3;
1189:         $$ = temp;
1190:     }
1191:     |   ACCURACY_SY EQUAL_SY STRING
1192:     {
1193:         variable* temp;
1194:         temp = new_variable();
1195:         temp->accuracy = $3;
1196:         $$ = temp;
1197:     }
1198:     |   SCALE_FACTOR_SY EQUAL_SY STRING
1199:     {
1200:         variable* temp;
1201:         temp = new_variable();
1202:         temp->scale_factor = $3;
1203:         $$ = temp;
1204:     }
1205:     |   SCALE_UNITS_SY EQUAL_SY STRING
1206:     {
1207:         variable* temp;
1208:         temp = new_variable();
1209:         temp->scale_units = $3;
1210:         $$ = temp;
1211:     }
1212:     |   R_LOW_SY EQUAL_SY STRING
1213:     {
1214:         variable* temp;
1215:         temp = new_variable();
1216:         temp->r_low = $3;
1217:         $$ = temp;
1218:     }
1219:     |   R_HIGH_SY EQUAL_SY STRING
1220:     {
1221:         variable* temp;
1222:         temp = new_variable();
1223:         temp->r_high = $3;

```

```

1224:         $$ = temp;
1225:     }
1226: |   Y_LOW_SY EQUAL_SY STRING
1227: {
1228:     variable* temp;
1229:     temp = new_variable();
1230:     temp->y_low = $3;
1231:     $$ = temp;
1232: }
1233: |   Y_HIGH_SY EQUAL_SY STRING
1234: {
1235:     variable* temp;
1236:     temp = new_variable();
1237:     temp->y_high = $3;
1238:     $$ = temp;
1239: }
1240: |
1241:     INVALID_VALUE_SY EQUAL_SY STRING
1242: {
1243:     variable* temp;
1244:     temp = new_variable();
1245:     temp->invalid_value = $3;
1246:     $$ = temp;
1247: }
1248: ;
1249:
1250: VAR_ELEMENTS      :   VAR_QUALIFIERS VAR_SUBELEMENTS
1251: {
1252:     $$ = merge_variables($1,$2);
1253: }
1254: ;
1255:
1256: VAR_QUALIFIERS    :   QUALIFIERS_SECTION
1257: {
1258:     variable* temp;
1259:     temp = new_variable();
1260:     temp->qualifiers = $1;
1261:     $$ = temp;
1262: }
1263: ;
1264:

```

```

1265:
1266: VAR_SUBELEMENTS      :   VAR_SUBELEMENTS VAR_SUBELEMENT
1267:      {
1268:          $$ = merge_variables($1,$2);
1269:      }
1270:      |   /*empty*/
1271:      {
1272:          $$ = NULL;
1273:      }
1274:      ;
1275:
1276: VAR_SUBELEMENT        :   DRANGE
1277:      {
1278:          variable* temp;
1279:          temp = new_variable();
1280:          temp->dranges = $1;
1281:          $$ = temp;
1282:      }
1283:      |   CURVE
1284:      {
1285:          variable* temp;
1286:          temp = new_variable();
1287:          temp->curves = $1;
1288:          $$ = temp;
1289:      }
1290:      |   LOCATION_SECTION
1291:      {
1292:          variable* temp;
1293:          temp = new_variable();
1294:          temp->location_data = $1;
1295:          $$ = temp;
1296:      }
1297:      |   ORIENTATION_SECTION
1298:      {
1299:          variable* temp;
1300:          temp = new_variable();
1301:          temp->orientation_data = $1;
1302:          $$ = temp;
1303:      }
1304:      ;
1305: //////////////////////////////////////

```

```

1306: //
1307: //      Variable references for command, fault, and data messages
1308: //
1309: //////////////////////////////////////
1310:
1311: VARIABLE_REFS      :   VARIABLE_REFS VARIABLE_REF
1312:      {
1313:          $$ = link_variable_ref($1,$2);
1314:      }
1315:      | /*empty*/
1316:      {
1317:          $$ = NULL;
1318:      }
1319:      ;
1320:
1321: VARIABLE_REF        :   OPEN_VARIABLE_REF_SY NAME_SY EQUAL_SY STRING
SLASHCLOSE_SY
1322:      {
1323:          $$ = new_variable_ref($4);
1324:      }
1325:      | OPEN_VARIABLE_REF_SY NAME_SY EQUAL_SY STRING CLOSE_SY
CLOSE_VARIABLE_REF_SY CLOSE_SY
1326:      {
1327:          $$ = new_variable_ref($4);
1328:      }
1329:      ;
1330: //////////////////////////////////////
1331: //
1332: //  Productions pertaining to discrete range variable definition in xTEDS
1333: //
1334: //////////////////////////////////////
1335:
1336: DRANGE              :   DRANGE_HEAD    DRANGE_OPTIONS    CLOSE_DRANGE_SY
CLOSE_SY
1337:      {
1338:          drange* temp;
1339:          temp = new_drange();
1340:          temp->options = $2;
1341:          $$ = merge_dranges($1,temp);
1342:      }
1343:      ;
1344:

```



```

1345: DRANGE_HEAD      :   OPEN_DRANGE_SY DRANGE_ATTRIBUTES CLOSE_SY
1346:      {
1347:          $$ = $2;
1348:      }
1349:      ;
1350:
1351: DRANGE_ATTRIBUTES   :   DRANGE_ATTRIBUTES DRANGE_ATTRIBUTE
1352:      {
1353:          $$ = merge_dranges($1,$2);
1354:      }
1355:      |   /*empty*/
1356:      {
1357:          $$ = NULL;
1358:      }
1359:      ;
1360:
1361: DRANGE_ATTRIBUTE :   NAME_SY EQUAL_SY STRING
1362:      {
1363:          drange* temp;
1364:          temp = new_drange();
1365:          temp->name = $3;
1366:          $$ = temp;
1367:      }
1368:      |   DESCRIPTION_SY EQUAL_SY STRING
1369:      {
1370:          drange* temp;
1371:          temp = new_drange();
1372:          temp->description = $3;
1373:          $$ = temp;
1374:      }
1375:      ;
1376:
1377: DRANGE_OPTIONS      :   DRANGE_OPTIONS DRANGE_OPTION
1378:      {
1379:          $$ = link_options($1,$2);
1380:      }
1381:      |   /*empty*/
1382:      {
1383:          $$ = NULL;
1384:      }
1385:      ;

```

```

1386:
1387: DRANGE_OPTION      :   OPEN_OPTION_SY OPTION_ATTRIBUTES SLASHCLOSE_SY
1388:      {
1389:          $$ = $2;
1390:      }
1391: |   OPEN_OPTION_SY OPTION_ATTRIBUTES CLOSE_SY CLOSE_OPTION_SY
CLOSE_SY
1392:      {
1393:          $$ = $2;
1394:      }
1395:      ;
1396:
1397: OPTION_ATTRIBUTES :   OPTION_ATTRIBUTES OPTION_ATTRIBUTE
1398:      {
1399:          $$ = merge_options($1,$2);
1400:      }
1401: |   /*empty*/
1402:      {
1403:          $$ = NULL;
1404:      }
1405:      ;
1406:
1407: OPTION_ATTRIBUTE  :   NAME_SY EQUAL_SY STRING
1408:      {
1409:          curveoption* temp;
1410:          temp = new_option();
1411:          temp->name = $3;
1412:          $$ = temp;
1413:      }
1414: |   VALUE_SY EQUAL_SY STRING
1415:      {
1416:          curveoption* temp;
1417:          temp = new_option();
1418:          temp->value = $3;
1419:          $$ = temp;
1420:      }
1421: |   DESCRIPTION_SY EQUAL_SY STRING
1422:      {
1423:          curveoption* temp;
1424:          temp = new_option();
1425:          temp->description = $3;

```

```

1426:         $$ = temp;
1427:     }
1428:     | ALARM_SY EQUAL_SY STRING
1429:     {
1430:         curveoption* temp;
1431:         temp = new_option();
1432:         temp->alarm = $3;
1433:         $$ = temp;
1434:     }
1435:     ;
1436: //////////////////////////////////////
1437: //
1438: //     Productions pertaining to curve variable definition in xTEDS
1439: //
1440: //////////////////////////////////////
1441:
1442: CURVE          :   CURVE_HEAD CURVE_COEFFS CLOSE_CURVE_SY CLOSE_SY
1443:     {
1444:         curve* temp;
1445:         temp = new_curve();
1446:         temp->coefs = $2;
1447:         $$ = merge_curves($1,temp);
1448:     }
1449:     ;
1450:
1451: CURVE_HEAD     :   OPEN_CURVE_SY CURVE_ATTRIBUTES CLOSE_SY
1452:     {
1453:         $$ = $2;
1454:     }
1455:     ;
1456:
1457: CURVE_ATTRIBUTES :   CURVE_ATTRIBUTES CURVE_ATTRIBUTE
1458:     {
1459:         $$ = merge_curves($1,$2);
1460:     }
1461:     | /*empty*/
1462:     {
1463:         $$ = NULL;
1464:     }
1465:     ;
1466:

```

```

1467: CURVE_ATTRIBUTE      :  NAME_SY EQUAL_SY STRING
1468:      {
1469:          curve* temp;
1470:          temp = new_curve();
1471:          temp->name = $3;
1472:          $$ = temp;
1473:      }
1474:      |  DESCRIPTION_SY EQUAL_SY STRING
1475:      {
1476:          curve* temp;
1477:          temp = new_curve();
1478:          temp->description = $3;
1479:          $$ = temp;
1480:      }
1481:      ;
1482:
1483: CURVE_COEFFS           :  CURVE_COEFFS CURVE_COEFF
1484:      {
1485:          $$ = link_coefs($1,$2);
1486:      }
1487:      |  /*empty*/
1488:      {
1489:          $$ = NULL;
1490:      }
1491:      ;
1492:
1493: CURVE_COEFF           :  OPEN_COEFF_SY COEFF_ATTRIBUTES SLASHCLOSE_SY
1494:      {
1495:          $$ = $2;
1496:      }
1497:      |  OPEN_COEFF_SY  COEFF_ATTRIBUTES  CLOSE_SY  CLOSE_COEFF_SY
CLOSE_SY
1498:      {
1499:          $$ = $2;
1500:      }
1501:      |  OPEN_COEFF_SY COEFF_ATTRIBUTES CLOSE_SY
1502:      {
1503:          $$ = $2;
1504:      }
1505:      ;
1506:

```

```

1507: COEFF_ATTRIBUTES : COEFF_ATTRIBUTES COEFF_ATTRIBUTE
1508:     {
1509:         $$ = merge_coefs($1,$2);
1510:     }
1511:     | /*empty*/
1512:     {
1513:         $$ = NULL;
1514:     }
1515:     ;
1516:
1517: COEFF_ATTRIBUTE      : EXPONENT_SY EQUAL_SY STRING
1518:     {
1519:         coef* temp;
1520:         temp = new_coef();
1521:         temp->exponent = $3;
1522:         $$ = temp;
1523:     }
1524:     | VALUE_SY EQUAL_SY STRING
1525:     {
1526:         coef* temp;
1527:         temp = new_coef();
1528:         temp->value = $3;
1529:         $$ = temp;
1530:     }
1531:     | DESCRIPTION_SY EQUAL_SY STRING
1532:     {
1533:         coef* temp;
1534:         temp = new_coef();
1535:         temp->description = $3;
1536:         $$ = temp;
1537:     }
1538:     ;
1539: %%

```

File: sdm/common/xTEDS/xTEDSVariable.cpp

```
1: #include "xTEDSVariable.h"
2: #include "MessageDef.h"
3: #include "../Exception/SDMBadIndexException.h"
4:
5: #include <string.h>
6: #include <stdlib.h>
7: #include <stdio.h>
8: #ifdef WIN32
9: # include "unistd.h"
10: #endif
11:
12: xTEDSVariable::xTEDSVariable():
13:   m_dataFormat(SDM_UINT16),
14:   m_iLength(1),
15:   m_strKind(NULL),
16:   m_strQualifier(NULL),
17:   m_iID(-1),
18:   m_strRangeMin(NULL),
19:   m_strRangeMax(NULL),
20:   m_strDefaultValue(NULL),
21:   m_iPrecision(ATTR_INIT_VALUE),
22:   m_strUnits(NULL),
23:   m_strAccuracy(NULL),
24:   m_strScaleFactor(NULL),
25:   m_strScaleUnits(NULL),
26:   m_varLocation(NULL),
27:   m_varOrientation(NULL),
28:   m_varDrange(NULL),
29:   m_varCurve(NULL),
30:   m_strRLow(NULL),
31:   m_strRHigh(NULL),
32:   m_strYLow(NULL),
33:   m_strYHigh(NULL),
34:   m_strInvalidValue(NULL)
35: {
36:   m_ItemType = TYPE_VARIABLE;
37:   //Initial invalidValue is NS for Not Specified
38:   m_strInvalidValue = strdup("NS");
39: }
```

```

40:
41: xTEDSVariable::~~xTEDSVariable()
42: {
43:     if(m_strKind!=NULL) free(m_strKind);
44:     if(m_strQualifier!=NULL) free(m_strQualifier);
45:     if(m_strDefaultValue!=NULL) free(m_strDefaultValue);
46:     if(m_strUnits!=NULL) free(m_strUnits);
47:     if(m_strScaleUnits!=NULL) free(m_strScaleUnits);
48:     if(m_strRangeMin != NULL) free(m_strRangeMin);
49:     if(m_strRangeMax != NULL) free(m_strRangeMax);
50:     if(m_strAccuracy != NULL) free(m_strAccuracy);
51:     if(m_strScaleFactor != NULL) free(m_strScaleFactor);
52:     if(m_strRLow != NULL) free(m_strRLow);
53:     if(m_strRHigh != NULL) free(m_strRHigh);
54:     if(m_strYLow != NULL) free(m_strYLow);
55:     if(m_strYHigh != NULL) free(m_strYHigh);
56:     if(m_strInvalidValue != NULL) free(m_strInvalidValue);
57:     if(m_varDrange!=NULL) delete(m_varDrange);
58:     if(m_varCurve!=NULL) delete(m_varCurve);
59:     if (m_varLocation != NULL) delete(m_varLocation);
60:     if (m_varOrientation != NULL) delete(m_varOrientation);
61: }
62:
63: void xTEDSVariable::SetVariable(const variable* NewVar)
64: {
65:     if (NewVar == NULL) return;
66:     if (NewVar->name      != NULL) setName(NewVar->name);
67:     if (NewVar->kind      != NULL) m_strKind = strdup(NewVar->kind);
68:     if (NewVar->interface_name != NULL) setInterfaceName(NewVar->interface_name);
69:     if (NewVar->interface_id  != NULL) setInterfaceID(atoi(NewVar->interface_id));
70:     if (NewVar->length       != NULL) m_iLength = atoi(NewVar->length);
71:     if (NewVar->qualifier    != NULL) m_strQualifier = strdup(NewVar->qualifier);
72:     if (NewVar->id          != NULL) m_iID = atoi(NewVar->id);
73:     if (NewVar->range_min    != NULL) m_strRangeMin = strdup(NewVar->range_min);
74:     if (NewVar->range_max    != NULL) m_strRangeMax = strdup(NewVar->range_max);
75:     if (NewVar->default_value != NULL) m_strDefaultValue = strdup(NewVar->default_value);
76:     if (NewVar->precision    != NULL) m_iPrecision = (unsigned int)atoi(NewVar->precision);
77:     if (NewVar->units        != NULL) m_strUnits = strdup(NewVar->units);
78:     if (NewVar->accuracy     != NULL) m_strAccuracy = strdup(NewVar->accuracy);
79:     if (NewVar->scale_factor != NULL) m_strScaleFactor = strdup(NewVar->scale_factor);
80:     if (NewVar->scale_units  != NULL) m_strScaleUnits = strdup(NewVar->scale_units);

```

```

81: if (NewVar->description != NULL) setDescription(NewVar->description);
82: if (NewVar->r_low != NULL) m_strRLow = strdup(NewVar->r_low);
83: if (NewVar->r_high != NULL) m_strRHigh = strdup(NewVar->r_high);
84: if (NewVar->y_low != NULL) m_strYLow = strdup(NewVar->y_low);
85: if (NewVar->y_high != NULL) m_strYHigh = strdup(NewVar->y_high);
86: if (NewVar->invalid_value != NULL) setInvalidValue(NewVar->invalid_value);
87:
88: if (NewVar->format != NULL)
89: {
90:   if( strcmp(NewVar->format,"INT08" )==0)
91:     m_dataFormat = SDM_INT08;
92:   else if(strcmp(NewVar->format,"UINT08" )==0)
93:     m_dataFormat = SDM_UINT08;
94:   else if(strcmp(NewVar->format,"INT16" )==0)
95:     m_dataFormat = SDM_INT16;
96:   else if(strcmp(NewVar->format,"UINT16" )==0)
97:     m_dataFormat = SDM_UINT16;
98:   else if(strcmp(NewVar->format,"INT32" )==0)
99:     m_dataFormat = SDM_INT32;
100:   else if(strcmp(NewVar->format,"UINT32" )==0)
101:     m_dataFormat = SDM_UINT32;
102:   else if(strcmp(NewVar->format,"FLOAT32")==0)
103:     m_dataFormat = SDM_FLOAT32;
104:   else if(strcmp(NewVar->format,"FLOAT64")==0)
105:     m_dataFormat = SDM_FLOAT64;
106: }
107:
108: if (NewVar->qualifiers != NULL)
109: {
110:   for (qualifier_type* CurQual = NewVar->qualifiers;
111:        CurQual != NULL;
112:        CurQual = CurQual->next)
113:     addQualifier (xTEDSQualifier(CurQual->name, CurQual->value, CurQual->units));
114: }
115:
116: if (NewVar->dranges != NULL) setDRange(NewVar->dranges);
117: if (NewVar->curves != NULL) setCurve(NewVar->curves);
118: if (NewVar->orientation_data != NULL) setOrientation(NewVar->orientation_data);
119: if (NewVar->location_data != NULL) setLocation(NewVar->location_data);
120:
121: }

```



```

122:
123: MessageDef* xTEDSVariable::RegInfo() const
124: {
125:     return NULL;
126: }
127:
128: void xTEDSVariable::VarRegInfo(char* InfoBufferOut, int BufferSize, int& start_byte) const
129: {
130:     char strBuf[MSG_DEF_SIZE];
131:     char strScale[32];
132:
133:     if(m_strScaleFactor == NULL)
134:         snprintf(strScale, sizeof(strScale), "1.0");
135:     else
136:         snprintf(strScale, sizeof(strScale), "%s", m_strScaleFactor);
137:
138:     switch(m_dataFormat)
139:     {
140:     case SDM_INT08:
141:         snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:INT08= \"%s \", start_byte, m_iLength,
strScale, m_strInvalidValue, m_strItemName);
142:         start_byte+=1*m_iLength;
143:         break;
144:     case SDM_UINT08:
145:         snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:UINT08= \"%s \", start_byte,
m_iLength, strScale, m_strInvalidValue, m_strItemName);
146:         start_byte+=1*m_iLength;
147:         break;
148:     case SDM_INT16:
149:         snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:INT16= \"%s \", start_byte, m_iLength,
strScale, m_strInvalidValue, m_strItemName);
150:         start_byte+=2*m_iLength;
151:         break;
152:     case SDM_UINT16:
153:         snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:UINT16= \"%s \", start_byte,
m_iLength, strScale, m_strInvalidValue, m_strItemName);
154:         start_byte+=2*m_iLength;
155:         break;
156:     case SDM_INT32:
157:         snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:INT32= \"%s \", start_byte, m_iLength,
strScale, m_strInvalidValue, m_strItemName);
158:         start_byte+=4*m_iLength;

```

```

159: break;
160: case SDM_UINT32:
161:     snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:UINT32= \"%s \", start_byte,
m_iLength, strScale, m_strInvalidValue, m_strItemName);
162:     start_byte+=4*m_iLength;
163:     break;
164: case SDM_FLOAT32:
165:     snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:FLOAT32= \"%s \", start_byte,
m_iLength, strScale, m_strInvalidValue, m_strItemName);
166:     start_byte+=4*m_iLength;
167:     break;
168: case SDM_FLOAT64:
169:     snprintf(strBuf, sizeof(strBuf), "Variable:%d:%d:%s:%s:FLOAT64= \"%s \", start_byte,
m_iLength, strScale, m_strInvalidValue, m_strItemName);
170:     start_byte+=8*m_iLength;
171:     break;
172: }
173: strncat(InfoBufferOut, strBuf, BufferSize - 1);
174: }
175:
176: void xTEDSVariable::VarRef(char* InfoBufferOut, int BufferSize) const
177: {
178:     char Buf[MSG_DEF_SIZE];
179:
180:     Buf[0] = '\0';
181:     snprintf(Buf, sizeof(Buf), "<VariableRef name= \"%s \"/>", m_strItemName);
182:
183:     strncat(InfoBufferOut, Buf, BufferSize - 1);
184: }
185:
186: VariableDef* xTEDSVariable::VarInfo() const
187: {
188:     char buf[MSG_DEF_SIZE] = "";
189:     char* temp = NULL;
190:     VariableDef* pVariableDef = new VariableDef();
191:
192:     temp = VarInfoRequest();
193:
194:     size_t str_length = strlen(temp);
195:     if (m_varDrange == NULL && m_varCurve == NULL && m_ItemQualifiers == NULL &&
m_varLocation == NULL)
196:     {

```

```

197:  if (str_length + 2 <= sizeof(buf))
198:  {
199:      strcpy(buf, temp);
200:      strcat(buf, ">");
201:  }
202: }
203: else
204: {
205:     size_t TempLength;
206:     if (str_length + 1 <= sizeof(buf))
207:     {
208:         strcpy(buf, temp);
209:         strcat(buf, ">");
210:         str_length += 1;
211:     }
212:     if (m_varDrange != NULL)
213:     {
214:         TempLength = strlen(buf);
215:         strncat(buf, " \n \t", sizeof(buf) - TempLength);
216:
217:         TempLength += 2;
218:         m_varDrange->VarInfoRequest(buf + TempLength, sizeof(buf) - TempLength);
219:     }
220:     else if (m_varCurve != NULL)
221:     {
222:         TempLength = strlen(buf);
223:         strncat(buf, " \n \t", sizeof(buf) - TempLength);
224:
225:         TempLength += 2;
226:         m_varCurve->VarInfoRequest(buf + TempLength, sizeof(buf) - TempLength);
227:     }
228:     if (m_ItemQualifiers != NULL)
229:     {
230:         TempLength = strlen(buf);
231:         strncat(buf, " \n \t", sizeof(buf) - TempLength);
232:
233:         TempLength += 2;
234:         m_ItemQualifiers->QualifierInfoRequest(buf + TempLength, sizeof(buf) - TempLength);
235:     }
236:     if (m_varLocation != NULL)
237:     {

```

```

238:     TempLength = strlen(buf);
239:     strncat(buf, " \n \t", sizeof(buf) - TempLength);
240:
241:     TempLength += 2;
242:     m_varLocation->VarInfoRequest(buf + TempLength, sizeof(buf) - TempLength);
243: }
244: if (m_varOrientation != NULL)
245: {
246:     TempLength = strlen(buf);
247:
248:     m_varOrientation->VarInfoRequest(buf + TempLength, sizeof(buf) - TempLength);
249: }
250: if (str_length + 12 <= sizeof(buf))
251: {
252:     strcat(buf, " \n</Variable>");
253: }
254: }
255: free (temp);
256:
257:
258: pVariableDef->SetDefinitions(buf);
259: pVariableDef->SetVariableName(m_strItemName);
260: pVariableDef->SetInterfaceName(m_strItemInterfaceName);
261:
262: return pVariableDef;
263: }
264:
265: bool xTEDSVariable::MatchesQualifier(const xTEDSQualifierList& QualList) const
266: {
267:     if(QualList.isEmpty())
268:         return true;
269:
270:     char id_str[8], format_str[8], length_str[8], precision_str[8];
271:     snprintf(length_str, sizeof(length_str), "%d", m_iLength);
272:     snprintf(id_str, sizeof(id_str), "%d", m_iID);
273:     snprintf(precision_str, sizeof(precision_str), "%u", m_iPrecision);
274:
275:     switch(m_dataFormat)
276:     {
277:     case SDM_INT08: strncpy(format_str, "INT08", sizeof(format_str)); break;
278:     case SDM_UINT08: strncpy(format_str, "UINT08", sizeof(format_str)); break;

```

```

279: case SDM_INT16: strncpy(format_str, "INT16", sizeof(format_str)); break;
280: case SDM_UINT16: strncpy(format_str, "UINT16", sizeof(format_str)); break;
281: case SDM_INT32: strncpy(format_str, "INT32", sizeof(format_str)); break;
282: case SDM_UINT32: strncpy(format_str, "UINT32", sizeof(format_str)); break;
283: case SDM_FLOAT32: strncpy(format_str, "FLOAT32", sizeof(format_str)); break;
284: case SDM_FLOAT64: strncpy(format_str, "FLOAT64", sizeof(format_str)); break;
285: }
286:
287: //check entire qualifier list
288: for (int i = 0; i < QualList.Size(); i++)
289: {
290:     try
291:     {
292:         const xTEDSQualifier& CurQual = QualList[i];
293:         if (CurQual.MatchesName("name"))
294:         {
295:             if (!CurQual.MatchesDescription(m_strItemName))
296:                 return false;
297:         }
298:         else if (CurQual.MatchesName("description"))
299:         {
300:             if (!CurQual.MatchesDescription(m_strItemDescription))
301:                 return false;
302:         }
303:         else if (CurQual.MatchesName("id"))
304:         {
305:             if (!CurQual.MatchesDescription(id_str))
306:                 return false;
307:         }
308:         else if (CurQual.MatchesName("length"))
309:         {
310:             if (!CurQual.MatchesDescription(length_str))
311:                 return false;
312:         }
313:         else if (CurQual.MatchesName("format"))
314:         {
315:             if (!CurQual.MatchesDescription(format_str))
316:                 return false;
317:         }
318:         else if (CurQual.MatchesName("kind"))
319:         {

```

```

320: if (!CurQual.MatchesDescription(m_strKind))
321:     return false;
322: }
323: else if (CurQual.MatchesName("rangeMin"))
324: {
325: if (!CurQual.MatchesDescription(m_strRangeMin))
326:     return false;
327: }
328: else if (CurQual.MatchesName("rangeMax"))
329: {
330: if (!CurQual.MatchesDescription(m_strRangeMax))
331:     return false;
332: }
333: else if (CurQual.MatchesName("defaultValue"))
334: {
335: if (!CurQual.MatchesDescription(m_strDefaultValue))
336:     return false;
337: }
338: else if (CurQual.MatchesName("m_iPrecision"))
339: {
340: if (!CurQual.MatchesDescription(precision_str))
341:     return false;
342: }
343: else if (CurQual.MatchesName("units"))
344: {
345: if (!CurQual.MatchesDescription(m_strUnits))
346:     return false;
347: }
348: else if (CurQual.MatchesName("accuracy"))
349: {
350: if (!CurQual.MatchesDescription(m_strAccuracy))
351:     return false;
352: }
353: else if (CurQual.MatchesName("scaleFactor"))
354: {
355: if (!CurQual.MatchesDescription(m_strScaleFactor))
356:     return false;
357: }
358: else if (CurQual.MatchesName("scaleUnits"))
359: {
360: if (!CurQual.MatchesDescription(m_strScaleUnits))

```

```

361:     return false;
362: }
363: else if (CurQual.MatchesName("rLow"))
364: {
365: if (!CurQual.MatchesDescription(m_strRLow))
366:     return false;
367: }
368: else if (CurQual.MatchesName("rHigh"))
369: {
370: if (!CurQual.MatchesDescription(m_strRHigh))
371:     return false;
372: }
373: else if (CurQual.MatchesName("yLow"))
374: {
375: if (!CurQual.MatchesDescription(m_strYLow))
376:     return false;
377: }
378: else if (CurQual.MatchesName("yHigh"))
379: {
380: if (!CurQual.MatchesDescription(m_strYHigh))
381:     return false;
382: }
383: else if (CurQual.MatchesName("invalidValue"))
384: {
385: if (!CurQual.MatchesDescription(m_strInvalidValue))
386:     return false;
387: }
388: else
389: return false;
390: }
391: catch (SDMBadIndexException& ex)
392: {
393:     printf("Error %s \n", ex.Message());
394:     return false;
395: }
396: }
397: return true;
398: }
399:
400: char* xTEDSVariable::VarInfoRequest() const
401: {

```

```

402: char Buf[MSG_DEF_SIZE] = "", TempBuf[512] = "";
403:
404: char strVarFormat[8];
405: getFormat(strVarFormat, sizeof(strVarFormat));
406:     snprintf(Buf, sizeof(Buf), "<Variable name= \"%s \" kind= \"%s \" format= \"%s \"",
m_strItemName, m_strKind, strVarFormat);
407:
408: if(m_strQualifier != NULL)
409: {
410:     snprintf(TempBuf, sizeof(TempBuf), " qualifier= \"%s \"", m_strQualifier);
411:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
412: }
413: if(m_iID != -1)
414: {
415:     snprintf(TempBuf, sizeof(TempBuf), " id= \"%d \"", m_iID);
416:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
417: }
418: if(m_strRangeMin != NULL)
419: {
420:     snprintf(TempBuf, sizeof(TempBuf), " rangeMin= \"%s \"", m_strRangeMin);
421:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
422: }
423: if(m_strRangeMax != NULL)
424: {
425:     snprintf(TempBuf, sizeof(TempBuf), " rangeMax= \"%s \"", m_strRangeMax);
426:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
427: }
428: if(m_strDefaultValue != NULL)
429: {
430:     snprintf(TempBuf, sizeof(TempBuf), " defaultValue= \"%s \"", m_strDefaultValue);
431:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
432: }
433: if(m_iPrecision != ATTR_INIT_VALUE)
434: {
435:     snprintf(TempBuf, sizeof(TempBuf), " m_iPrecision= \"%u \"", m_iPrecision);
436:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
437: }
438: if(m_strUnits != NULL)
439: {
440:     snprintf(TempBuf, sizeof(TempBuf), " units= \"%s \"", m_strUnits);
441:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));

```



```

442: }
443: if(m_strAccuracy != NULL)
444: {
445:     snprintf(TempBuf, sizeof(TempBuf), " accuracy= \"%s \", m_strAccuracy);
446:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
447: }
448: if(m_strScaleFactor != NULL)
449: {
450:     snprintf(TempBuf, sizeof(TempBuf), " scaleFactor= \"%s \", m_strScaleFactor);
451:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
452: }
453: if(m_strScaleUnits != NULL)
454: {
455:     snprintf(TempBuf, sizeof(TempBuf), " scaleUnits= \"%s \", m_strScaleUnits);
456:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
457: }
458: if(m_iLength > 1)
459: {
460:     snprintf(TempBuf, sizeof(TempBuf), " length= \"%d \", m_iLength);
461:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
462: }
463: if(m_strItemDescription!=NULL)
464: {
465:     snprintf(TempBuf, sizeof(TempBuf), " description= \"%s \", m_strItemDescription);
466:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
467: }
468: if(m_strRLow != NULL)
469: {
470:     snprintf(TempBuf, sizeof(TempBuf), " rLow= \"%s \", m_strRLow);
471:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
472: }
473: if(m_strRHigh != NULL)
474: {
475:     snprintf(TempBuf, sizeof(TempBuf), " rHigh= \"%s \", m_strRHigh);
476:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
477: }
478: if(m_strYLow != NULL)
479: {
480:     snprintf(TempBuf, sizeof(TempBuf), " yLow= \"%s \", m_strYLow);
481:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
482: }

```

```

483: if(m_strYHigh != NULL)
484: {
485:     snprintf(TempBuf, sizeof(TempBuf), " yHigh= \"%s \"", m_strYHigh);
486:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
487: }
488: if(m_strInvalidValue!=NULL && strcmp(m_strInvalidValue,"NS")!=0)
489: {
490:     snprintf(TempBuf, sizeof(TempBuf), " invalidValue= \"%s \"", m_strInvalidValue);
491:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
492: }
493: return strdup(Buf);
494: }
495:
496: void xTEDSVariable::getFormat(char* strFormatOut, int BufSize) const
497: {
498:     if (BufSize < 8)
499:         return ;
500:
501:     switch(m_dataFormat)
502:     {
503:     case SDM_INT08:
504:         sprintf(strFormatOut,"INT08");
505:         break;
506:     case SDM_UINT08:
507:         sprintf(strFormatOut,"UINT08");
508:         break;
509:     case SDM_INT16:
510:         sprintf(strFormatOut,"INT16");
511:         break;
512:     case SDM_UINT16:
513:         sprintf(strFormatOut,"UINT16");
514:         break;
515:     case SDM_INT32:
516:         sprintf(strFormatOut,"INT32");
517:         break;
518:     case SDM_UINT32:
519:         sprintf(strFormatOut,"UINT32");
520:         break;
521:     case SDM_FLOAT32:
522:         sprintf(strFormatOut,"FLOAT32");
523:         break;

```

```

524: case SDM_FLOAT64:
525:     sprintf(strFormatOut,"FLOAT64");
526:     break;
527: }
528: }
529:
530: void xTEDSVariable::setLocation(const location *loc)
531: {
532:     if (loc == NULL) return;
533:     if (m_varLocation == NULL)
534:         m_varLocation = new xTEDSLocation();
535:     m_varLocation->setLocation(loc);
536: }
537:
538: void xTEDSVariable::setOrientation(const orientation *orient)
539: {
540:     if (orient == NULL) return;
541:     if (m_varOrientation == NULL)
542:         m_varOrientation = new xTEDSOrientationList();
543:     m_varOrientation->SetOrientation(orient);
544: }
545:
546: void xTEDSVariable::setDRange(const drange* range)
547: {
548:     if (range == NULL) return;
549:     if (m_varDrange == NULL)
550:         m_varDrange = new xTEDSDrange();
551:     m_varDrange->setDRange(range);
552: }
553:
554: void xTEDSVariable::setCurve(const curve* curves)
555: {
556:     if (curves == NULL) return;
557:     if (m_varCurve == NULL)
558:         m_varCurve = new xTEDSCurve();
559:     m_varCurve->setCurve(curves);
560: }
561:
562: bool xTEDSVariable::RegInfoMatch(const char* pname, const xTEDSQualifierList& qualifiers,
const char* interface) const
563: {

```

```

564: if (pname == NULL || strcmp(m_strItemName, pname)==0)
565: {
566:   if (interface == NULL || strcmp(m_strItemInterfaceName, interface)==0)
567:   {
568:     if (MatchesQualifier(qualifiers))
569:       return true;
570:   }
571: }
572: return false;
573: }
574:
575: #ifndef REMOVE_DEBUG_OUTPUT
576: void xTEDSVariable::PrintDebug() const
577: {
578:   printf("  ");
579:   xTEDSItem::PrintDebug();
580:   char* VariableStr = VarInfoRequest();
581:   printf("    %s/> \n", VariableStr);
582:   printf(" \n");
583:   free(VariableStr);
584: }
585: #endif
586:
587: //-----
588: // Getter/Setter Methods added for testability
589: //-----
590:
591: void xTEDSVariable::setStringMember( char* & memberStrVar, const char* newValue )
592: {
593:   if (newValue == NULL)
594:     return;
595:
596:   if (memberStrVar)
597:   {
598:     free(memberStrVar);
599:   }
600:
601:   memberStrVar = strdup(newValue);
602: }
603:
604: /**

```

```

605: * setter for data format
606: *
607: * @param newDataFormat - new value for data format
608: */
609: void xTEDSVVariable::setDataFormat( SDMDDataTypes& newDataFormat )
610: {
611:     m_dataFormat = newDataFormat;
612: }
613:
614: const SDMDDataTypes& xTEDSVVariable::getDataFormat() const
615: {
616:     return m_dataFormat;
617: }
618:
619: void xTEDSVVariable::setLength( int newLength )
620: {
621:     m_iLength = newLength;
622: }
623:
624: int xTEDSVVariable::getLength() const
625: {
626:     return m_iLength;
627: }
628:
629: void xTEDSVVariable::setKind( const char* newKind )
630: {
631:     setStringMember(m_strKind, newKind);
632: }
633:
634: const char* xTEDSVVariable::getKind() const
635: {
636:     return m_strKind;
637: }
638:
639: void xTEDSVVariable::setQualifier( const char* newQualifier )
640: {
641:     setStringMember(m_strQualifier, newQualifier);
642: }
643:
644: const char* xTEDSVVariable::getQualifier() const
645: {

```

```

646: return m_strQualifier;
647: }
648:
649: void xTEDSVariable::setID( int newID )
650: {
651:     m_iID = newID;
652: }
653:
654: int xTEDSVariable::getID() const
655: {
656:     return m_iID;
657: }
658:
659: void xTEDSVariable::setRangeMin( const char* newRangeMin )
660: {
661:     setStringMember(m_strRangeMin, newRangeMin);
662: }
663:
664: const char* xTEDSVariable::getRangeMin() const
665: {
666:     return m_strRangeMin;
667: }
668:
669: void xTEDSVariable::setRangeMax( const char* newRangeMax )
670: {
671:     setStringMember(m_strRangeMax, newRangeMax);
672: }
673:
674: void xTEDSVariable::setDefaultValue( const char* newDefaultValue )
675: {
676:     setStringMember(m_strDefaultValue, newDefaultValue);
677: }
678:
679: const char* xTEDSVariable::getDefaultValue() const
680: {
681:     return m_strDefaultValue;
682: }
683:
684: void xTEDSVariable::setPrecision( unsigned int newPrecesion )
685: {
686:     m_iPrecision = newPrecesion;

```

```

687: }
688:
689: unsigned int xTEDSVariable::getPrecision() const
690: {
691:     return m_iPrecision;
692: }
693:
694: void xTEDSVariable::setUnits( const char* newUnits )
695: {
696:     setStringMember(m_strUnits, newUnits);
697: }
698:
699: const char* xTEDSVariable::getUnits() const
700: {
701:     return m_strUnits;
702: }
703:
704: void xTEDSVariable::setAccuracy( const char* newAccuracy )
705: {
706:     setStringMember(m_strAccuracy, newAccuracy);
707: }
708:
709: const char* xTEDSVariable::getAccuracy() const
710: {
711:     return m_strAccuracy;
712: }
713:
714: void xTEDSVariable::setScaleFactor( const char* newScaleFactor )
715: {
716:     setStringMember(m_strScaleFactor, newScaleFactor);
717: }
718:
719: const char* xTEDSVariable::getScaleFactor() const
720: {
721:     return m_strScaleFactor;
722: }
723:
724: void xTEDSVariable::setScaleUnits( const char* newScaleUnits )
725: {
726:     setStringMember(m_strScaleUnits, newScaleUnits);
727: }

```

```

728:
729: const char* xTEDSVariable::getScaleUnits() const
730: {
731:     return m_strScaleUnits;
732: }
733:
734: void xTEDSVariable::setRLow( const char* newRLow )
735: {
736:     setStringMember( m_strRLow, newRLow );
737: }
738:
739: const char* xTEDSVariable::getRLow() const
740: {
741:     return m_strRLow;
742: }
743:
744: void xTEDSVariable::setRHigh( const char* newRHigh )
745: {
746:     setStringMember( m_strRHigh, newRHigh );
747: }
748:
749: const char* xTEDSVariable::getRHigh() const
750: {
751:     return m_strRHigh;
752: }
753:
754: void xTEDSVariable::setYLow( const char* newYLow )
755: {
756:     setStringMember( m_strYLow, newYLow );
757: }
758:
759: const char* xTEDSVariable::getYLow() const
760: {
761:     return m_strYLow;
762: }
763:
764: void xTEDSVariable::setYHigh( const char* newYHigh )
765: {
766:     setStringMember( m_strYHigh, newYHigh );
767: }
768:

```



```
769: const char* xTEDSVariable::getYHigh() const
770: {
771:     return m_strYHigh;
772: }
773:
774: void xTEDSVariable::setInvalidValue(const char* newInvalidValue)
775: {
776:     setStringMember(m_strInvalidValue, newInvalidValue);
777: }
778:
779: const char* xTEDSVariable::getInvalidValue() const
780: {
781:     return m_strInvalidValue;
782: }
783:
784:
785:
786:
787:
788:
789:
```

File: sdm/common/xTEDS/xTEDSOptionList.cpp

```
1: #include "xTEDSOptionList.h"
2: #include "MessageDef.h"
3:
4: #include <stdlib.h>
5: #include <string.h>
6: #include <stdio.h>
7:
8: /* Not used..
9: xTEDSOptionList::xTEDSOptionList(const xTEDSOptionList& b):head(NULL),tail(NULL)
10: {
11:     head = copyList(b.head,&tail);
12: }
13:
14: xTEDSOptionList& xTEDSOptionList::operator=(const xTEDSOptionList& b)
15: {
16:     deleteList(head);
17:     head = copyList(b.head,&tail);
18:     return *this;
19: }
20:     struct    xTEDSOptionListNode*    copyList(struct    xTEDSOptionListNode*    list,struct
xTEDSOptionListNode** tail)
21: {
22:     struct xTEDSOptionListNode* p;
23:     struct xTEDSOptionListNode* head;
24:     head = (struct xTEDSOptionListNode*)malloc(sizeof(struct xTEDSOptionListNode));
25:     p = head;
26:     for(struct xTEDSOptionListNode* cur=list;cur!=NULL;cur=cur->next)
27:     {
28:         p->data = cur->data;
29:         if(cur->next!=NULL)
30:         {
31:             p->next = (struct xTEDSOptionListNode*)malloc(sizeof(struct xTEDSOptionListNode));
32:         }
33:         else
34:         {
35:             p->next = NULL;
36:             *tail = p;
37:         }
38:         p = p->next;
```

```

39: }
40: return head;
41: }*/
42:
43: void deleteList(struct xTEDSOOptionListNode* p)
44: {
45: for (xTEDSOOptionListNode *Cur = p; Cur != NULL; )
46: {
47:     xTEDSOOptionListNode *Temp = Cur->next;
48:     delete Cur;
49:     Cur = Temp;
50: }
51: }
52:
53: xTEDSOOptionList::xTEDSOOptionList():head(NULL),tail(NULL)
54: {}
55:
56:
57: xTEDSOOptionList::~~xTEDSOOptionList()
58: {
59: deleteList(head);
60: }
61:
62: void xTEDSOOptionList::setOptionsList(const curveoption* OptionList)
63: {
64: addOption(OptionList);
65: for (curveoption* Cur = OptionList->next; Cur != NULL; Cur = Cur->next)
66:     addOption(Cur);
67: }
68:
69: void xTEDSOOptionList::addOption(const curveoption* NewOption)
70: {
71: xTEDSOOptionListNode *NewNode = new xTEDSOOptionListNode();
72: NewNode->data.setOption(NewOption);
73: if (head == NULL)
74: {
75:     head = tail = NewNode;
76: }
77: else
78: {
79:     xTEDSOOptionListNode *Cur;

```

```

80:     for (Cur = head; Cur->next != NULL; Cur = Cur->next)
81:         ;
82:     Cur->next = NewNode;
83:     tail = NewNode;
84: }
85: }
86:
87: bool xTEDSOptionList::IsEmpty() const
88: {
89:     if(head == NULL)
90:         return true;
91:     return false;
92: }
93:
94: void xTEDSOptionList::VarInfoRequest( char* InfoBufferOut, size_t BufferSize ) const
95: {
96:     const unsigned int MAX_BUF_SIZE = 1024;
97:     char Buf[MAX_BUF_SIZE];
98:     if (head == NULL)
99:         return ;
100:
101:     Buf[0] = '\0';
102:     for (xTEDSOptionListNode *Cur = head; Cur != NULL; Cur = Cur->next)
103:     {
104:         strncat(Buf, "\n \t \t", sizeof(Buf) - strlen(Buf));
105:
106:         const size_t CurBufLength = strlen(Buf);
107:         Cur->data.VarInfoRequest(Buf + CurBufLength, sizeof(Buf) - CurBufLength);
108:     }
109:     strncat(InfoBufferOut, Buf, BufferSize - 1);
110: }

```

File: sdm/common/xTEDS/xTEDSCoef.cpp

```
1: #include "xTEDSCoef.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10: xTEDSCoef::xTEDSCoef():m_strDescription(NULL),m_strValue(NULL),m_strExponent(NULL)
11: {
12: }
13:
14: /*(xTEDSCoef::xTEDSCoef(const xTEDSCoef&
b):m_strDescription(NULL),m_strValue(NULL),m_strExponent(NULL)
15: {
16: if(m_strDescription!=NULL) free(m_strDescription);
17: m_strDescription = strdup(b.m_strDescription);
18: if(m_strValue!=NULL) free(m_strValue);
19: m_strValue = strdup(b.m_strValue);
20: if(m_strExponent!=NULL) free(m_strExponent);
21: m_strExponent = strdup(b.m_strExponent);
22: }
23: xTEDSCoef& xTEDSCoef::operator=(const xTEDSCoef& b)
24: {
25: if(m_strDescription!=NULL) free(m_strDescription);
26: m_strDescription = strdup(b.m_strDescription);
27: if(m_strValue!=NULL) free(m_strValue);
28: m_strValue = strdup(b.m_strValue);
29: if(m_strExponent!=NULL) free(m_strExponent);
30: m_strExponent = strdup(b.m_strExponent);
31: return *this;
32: }*/
33:
34: xTEDSCoef::~xTEDSCoef()
35: {
36: if(m_strDescription!=NULL) free(m_strDescription);
37: if(m_strValue!=NULL) free(m_strValue);
38: if(m_strExponent!=NULL) free(m_strExponent);
```

```

39: }
40:
41: void xTEDSCoef::setDescription(const char* new_description)
42: {
43: if (new_description == NULL) return;
44: if(m_strDescription!=NULL) free(m_strDescription);
45: m_strDescription = strdup(new_description);
46: }
47:
48: void xTEDSCoef::setValue(const char* new_value)
49: {
50: if (new_value == NULL) return;
51: if(m_strValue!=NULL) free(m_strValue);
52: m_strValue = strdup(new_value);
53: }
54:
55: void xTEDSCoef::setExponent(const char* new_exponent)
56: {
57: if (new_exponent == NULL) return;
58: if(m_strExponent!=NULL) free(m_strExponent);
59: m_strExponent = strdup(new_exponent);
60: }
61:
62: void xTEDSCoef::setCoef(const coef* NewCoef)
63: {
64: if (NewCoef == NULL) return;
65:
66: setExponent(NewCoef->exponent);
67: setValue(NewCoef->value);
68: setDescription(NewCoef->description);
69: }
70:
71: void xTEDSCoef::VarInfoRequest( char* InfoBufferOut, size_t BufferSize ) const
72: {
73: const unsigned int MAX_BUF_SIZE = 512;
74: char Buf[MAX_BUF_SIZE], TempBuf[MAX_BUF_SIZE];
75:
76: if (m_strExponent == NULL || m_strValue == NULL) return ;
77:
78: snprintf(Buf, sizeof(Buf), "<Coef exponent= \"%s \" value= \"%s \"\",m_strExponent,m_strValue);
79:

```

```
80: if(m_strDescription!=NULL)
81: {
82:     snprintf(TempBuf, sizeof(TempBuf), " description= \"%s \"/>", m_strDescription);
83:
84:     strncat(Buf, TempBuf, sizeof(Buf) - strlen(Buf));
85: }
86: else
87:     strncat(Buf, "/>", sizeof(Buf) - strlen(Buf));
88:
89: strncat(InfoBufferOut, Buf, BufferSize - 1);
90: }
91:
```

File: sdm/common/xTEDS/MessageDef.cpp

```
1: #include "MessageDef.h"
2:
3: #include <stdlib.h>
4: #include <stdio.h>
5: #include <string.h>
6:
7: MessageDef::MessageDef():mMessageInterfaceID(),def(NULL),xTEDSPortion(NULL),next(NULL)
8: {}
9:
10: MessageDef::MessageDef(const MessageDef&
b):mMessageInterfaceID(b.mMessageInterfaceID),def(b.def),xTEDSPortion(b.xTEDSPortion),next(b.ne
xt)
11: {}
12:
13: MessageDef::~MessageDef()
14: {
15: if(def != NULL)
16: {
17:     free(def);
18: }
19: if(xTEDSPortion != NULL)
20: {
21:     free(xTEDSPortion);
22: }
23: if(next != NULL)
24: {
25:     delete next;
26: }
27: }
28:
29: MessageDef& MessageDef::operator=(const MessageDef& b)
30: {
31: def = b.def;
32: xTEDSPortion = b.xTEDSPortion;
33: next = b.next;
34: return *this;
35: }
36:
37: void MessageDef::operator += (MessageDef* Right)
```



```

38: {
39: Join(Right);
40: }
41:
42: void MessageDef::Join(MessageDef* b)
43: {
44: //join the two lists
45: MessageDef* cur;
46: if(b == NULL)
47:     return;
48: if(next == NULL)
49: {
50:     next = b;
51:     return;
52: }
53: for(cur = next;cur->next!=NULL;cur=cur->next);
54: cur->next = b;
55: }
56:
57: void MessageDef::SetInterfaceMessageID(const SDMMMessage_ID& NewID)
58: {
59: mMessageInterfaceID = NewID;
60: }
61:
62: void MessageDef::SetDefinitions(const char* NewDef)
63: {
64: if (NewDef == NULL) return;
65: if (def != NULL) free (def);
66: def = strdup(NewDef);
67: }
68:
69: void MessageDef::SetxTEDSPortion(const char* NewxTEDS)
70: {
71: if (NewxTEDS == NULL) return;
72: if (xTEDSPortion != NULL) free(xTEDSPortion);
73: xTEDSPortion = strdup(NewxTEDS);
74: }
75:
76: void MessageDef::Print()
77: {
78: printf("msg_def: %s message: %s \n",def,xTEDSPortion);

```

```
79: if(next!=NULL)
80:     next->Print();
81: }
```

File: sdm/common/xTEDS/xTEDSCommand.cpp

```
1: #include "xTEDSCommand.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #ifdef WIN32
7: # include "unistd.h"
8: #endif
9:
10: xTEDSCommand::xTEDSCommand():m_CommandMessage(NULL),m_FaultMessage(NULL)
11: {
12:     Type = WRAPPER_COMMAND;
13: }
14:
15: xTEDSCommand::~~xTEDSCommand()
16: {
17:     if (m_CommandMessage != NULL)
18:         delete m_CommandMessage;
19:     if (m_FaultMessage != NULL)
20:         delete m_FaultMessage;
21: }
22:
23: bool xTEDSCommand::SetCommand(const command* Command, const xTEDSVariableList&
VariableList)
24: {
25:     if (Command == NULL || Command->command_message == NULL)
26:         return false;
27:
28:     // Set the command attributes
29:     if (Command->interface_name != NULL) setInterfaceName (Command->interface_name);
30:     if (Command->interface_id != NULL) setInterfaceID (atoi(Command->interface_id));
31:
32:     // Set the command message, required
33:     m_CommandMessage = new xTEDSCommandMsg();
34:     bool ReturnResult = m_CommandMessage->SetCommandMsg(Command->command_message,
VariableList);
35:     if (ReturnResult == false)
36:         return false;
37:
```

```

38: // Set the fault message, optional
39: if (Command->fault_message != NULL)
40: {
41:     m_FaultMessage = new xTEDSFaultMsg();
42:     ReturnResult = m_FaultMessage->SetFaultMsg(Command->fault_message, VariableList);
43: }
44: return ReturnResult;
45: }
46:
47: MessageDef* xTEDSCommand::RegInfo(const char* ItemName) const
48: {
49: char DefinitionsBuf[MSG_DEF_SIZE] = "";
50: char xTEDSPortionBuf[MSG_DEF_SIZE] = "";
51: char VarPortionBuf[MSG_DEF_SIZE] = "";
52:
53: if (m_CommandMessage == NULL) return NULL;
54:
55: MessageDef* pCmdMsgDef = m_CommandMessage->RegInfo();
56: if (NULL == pCmdMsgDef)
57:     return NULL;
58:
59: VariableDef* pCmdVarDef = m_CommandMessage->GetVariableXtedsDefinitions();
60:
61: if(m_FaultMessage!=NULL)    // If there exists a fault message
62: {
63:     MessageDef* pFltMsgDef = m_FaultMessage->RegInfo();
64:     if (NULL == pFltMsgDef)
65:     {
66:         free (pCmdMsgDef);
67:         free (pCmdVarDef);
68:         return NULL;
69:     }
70:     VariableDef* pFltVarDef = m_FaultMessage->GetVariableXtedsDefinitions();
71:
72:     snprintf(DefinitionsBuf,sizeof(DefinitionsBuf),"<Command> \n \t%s \n \t%s \n</Command>",
73:         pCmdMsgDef->GetDefinitions(),
74:         pFltMsgDef->GetDefinitions());
75:
76:     // Get the full variable xTEDS definitions, into VarPortionBuf
77:     VariableDef::ToStringConcatVariableDefsIgnoreDuplicates ( VarPortionBuf,
78:         sizeof(VarPortionBuf), pCmdVarDef, pFltVarDef );

```

```

79:
80:     snprintf(xTEDSPortionBuf,sizeof(xTEDSPortionBuf),"%s \n<Command> \n \t%s \n \t%s
\n</Command>",
81:         VarPortionBuf, pCmdMsgDef->GetxTEDSPortion(), pFltMsgDef->GetxTEDSPortion());
82:
83:     delete pFltMsgDef;
84:     pFltMsgDef = NULL;
85:
86:     delete pFltVarDef;
87:     pFltVarDef = NULL;
88: }
89: else          // If this is only a command message
90: {
91:     snprintf(DefinitionsBuf,sizeof(DefinitionsBuf),"<Command> \n \t%s \n</Command>",
92:         pCmdMsgDef->GetDefinitions());
93:
94:     VariableDef::ToStringConcatVariableDefs(VarPortionBuf, sizeof(VarPortionBuf),
95:         pCmdVarDef, NULL);
96:
97:     snprintf(xTEDSPortionBuf,sizeof(xTEDSPortionBuf),"%s \n<Command> \n \t%s
\n</Command>",
98:         VarPortionBuf, pCmdMsgDef->GetxTEDSPortion());
99: }
100:
101: MessageDef* ReturnDef = new MessageDef();
102:
103: //
104: // Set the message id to the id of the requested item
105: ReturnDef->SetInterfaceMessageID(pCmdMsgDef->GetInterfaceMessageID());
106: if (ItemName != NULL)
107: {
108:     if (m_FaultMessage != NULL && m_FaultMessage->NameEquals(ItemName))
109:         ReturnDef->SetInterfaceMessageID(m_FaultMessage->GetID());
110: }
111:
112: delete(pCmdMsgDef);
113: if (NULL != pCmdVarDef)
114:     delete pCmdVarDef;
115:
116: ReturnDef->SetDefinitions(DefinitionsBuf);
117: ReturnDef->SetxTEDSPortion(xTEDSPortionBuf);

```

```

118:   return ReturnDef;
119: }
120:
121: bool xTEDSCommand::RegInfoMatch(const char* Name, const xTEDSQualifierList& Qualifiers,
const char* Interface) const
122: {
123:   if (m_CommandMessage != NULL && m_CommandMessage->RegInfoMatch(Name,
Qualifiers, Interface))
124:     return true;
125:   else if (m_FaultMessage != NULL && m_FaultMessage->RegInfoMatch(Name, Qualifiers,
Interface))
126:     return true;
127:
128:   return false;
129: }
130:
131: bool xTEDSCommand::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList,
const char* Interface) const
132: {
133:   if (m_CommandMessage != NULL && m_CommandMessage->RegexMatch(Pattern, QualList,
Interface))
134:     return true;
135:   else if (m_FaultMessage != NULL && m_FaultMessage->RegexMatch(Pattern, QualList,
Interface))
136:     return true;
137:
138:   return false;
139: }
140:
141: bool xTEDSCommand::ContainsMessage(const SDMMMessage_ID& MessageID) const
142: {
143:   if (m_CommandMessage != NULL && m_CommandMessage->GetID() == MessageID)
144:     return true;
145:   else if (m_FaultMessage != NULL && m_FaultMessage->GetID() == MessageID)
146:     return true;
147:   return false;
148: }
149:
150: SDMMMessage_ID xTEDSCommand::GetFaultMessageID() const
151: {
152:   if (m_FaultMessage != NULL)
153:     return m_FaultMessage->GetID();

```

```
154:    return SDMMMessage_ID('\0', '\0');
155: }
156:
157: #ifndef REMOVE_DEBUG_OUTPUT
158: void xTEDSCommand::PrintDebug() const
159: {
160:     xTEDSWrapper::PrintDebug();
161:     printf(" xTEDSCommand \n");
162:
163:     if (m_CommandMessage!=NULL)
164:         m_CommandMessage->PrintDebug();
165:
166:     if (m_FaultMessage!=NULL)
167:         m_FaultMessage->PrintDebug();
168: }
169: #endif
```

File: sdm/common/xTEDS/xTEDSItem.cpp

```
1: #include "xTEDSItem.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include "../Regex/Regex.h"
6:
7:
xTEDSItem::xTEDSItem():m_strItemName(NULL),m_strItemDescription(NULL),m_ItemQualifiers(NULL),m_ItemType(TYPE_NONE),m_strItemInterfaceName(NULL),m_ItemInterfaceMessageID()
8: {
9: }
10:
11: xTEDSItem::xTEDSItem(const xTEDSItem&
b):m_strItemName(NULL),m_strItemDescription(NULL),m_ItemQualifiers(b.m_ItemQualifiers),m_ItemType(b.m_ItemType),m_strItemInterfaceName(NULL),m_ItemInterfaceMessageID(b.m_ItemInterfaceMessageID)
12: {
13: if(m_strItemName!=NULL) free(m_strItemName);
14: m_strItemName = strdup(b.m_strItemName);
15: if(m_strItemDescription!=NULL) free(m_strItemDescription);
16: m_strItemDescription = strdup(b.m_strItemDescription);
17: if(m_strItemInterfaceName!=NULL) free(m_strItemInterfaceName);
18: m_strItemInterfaceName = strdup(b.m_strItemInterfaceName);
19: }
20:
21: xTEDSItem::~~xTEDSItem()
22: {
23: if(m_strItemName!=NULL) free(m_strItemName);
24: if(m_strItemDescription!=NULL) free(m_strItemDescription);
25: if(m_strItemInterfaceName!=NULL) free(m_strItemInterfaceName);
26: if(m_ItemQualifiers!=NULL) delete(m_ItemQualifiers);
27: }
28:
29: void xTEDSItem::setName(char* new_name)
30: {
31: if (new_name == NULL) return;
32: if(m_strItemName!=NULL) free(m_strItemName);
33: m_strItemName = strdup(new_name);
34: }
35:
```



```

36: void xTEDSItem::setDescription(char* new_description)
37: {
38: if (new_description == NULL) return;
39: if(m_strItemDescription!=NULL) free(m_strItemDescription);
40: m_strItemDescription = strdup(new_description);
41: }
42:
43: void xTEDSItem::setInterfaceName(char* new_interfacename)
44: {
45: if (new_interfacename == NULL) return;
46: if(m_strItemInterfaceName!=NULL) free(m_strItemInterfaceName);
47: m_strItemInterfaceName = strdup(new_interfacename);
48: }
49:
50: void xTEDSItem::setInterfaceID(int id)
51: {
52: if (id > 0 && id < 255)
53:     m_ItemInterfaceMessageID.setInterface(static_cast<unsigned char>(id));
54: }
55:
56: xTEDSItem& xTEDSItem::operator=(const xTEDSItem& b)
57: {
58: if(m_strItemName!=NULL) free(m_strItemName);
59: m_strItemName = strdup(b.m_strItemName);
60: if(m_strItemDescription!=NULL) free(m_strItemDescription);
61: m_strItemDescription = strdup(b.m_strItemDescription);
62: if(m_strItemInterfaceName!=NULL) free(m_strItemInterfaceName);
63: m_strItemInterfaceName = strdup(b.m_strItemInterfaceName);
64: m_ItemType = b.m_ItemType;
65: m_ItemInterfaceMessageID = b.m_ItemInterfaceMessageID;
66: m_ItemQualifiers = b.m_ItemQualifiers;
67: return *this;
68: }
69:
70: void xTEDSItem::addQualifier(const xTEDSQualifier &NewQualifier)
71: {
72: if (m_ItemQualifiers == NULL)
73:     m_ItemQualifiers = new xTEDSQualifierList();
74: m_ItemQualifiers->addQualifier(NewQualifier);
75: }
76:

```

```

77: bool xTEDSItem::NameEquals(const char* CompareName) const
78: {
79: if (strcmp(m_strItemName, CompareName) == 0)
80:     return true;
81: return false;
82: }
83:
84: bool xTEDSItem::RegexMatch(const char* Pattern, const xTEDSQualifierList& QualList, const
char* Interface) const
85: {
86: if (DoesRegexMatch(m_strItemName, Pattern))
87:     return true;
88: return false;
89: }
90:
91: #ifndef REMOVE_DEBUG_OUTPUT
92: void xTEDSItem::PrintDebug() const
93: {
94: char IDStr[128];
95: char ItemTypeStr[128];
96:
97: switch(m_ItemType)
98: {
99:     case TYPE_VARIABLE:
100:     strcpy(ItemTypeStr, "TYPE_VARIABLE");
101:     break;
102:     case TYPE_DATAMSG:
103:     strcpy(ItemTypeStr, "TYPE_DATAMSG");
104:     break;
105:     case TYPE_COMMANDMSG:
106:     strcpy(ItemTypeStr, "TYPE_COMMANDMSG");
107:     break;
108:     case TYPE_FAULTMSG:
109:     strcpy(ItemTypeStr, "TYPE_FAULTMSG");
110:     break;
111:     case TYPE_DATAREPLYMSG:
112:     strcpy(ItemTypeStr, "TYPE_DATAREPLYMSG");
113:     break;
114:     case TYPE_NONE:
115:     strcpy(ItemTypeStr, "TYPE_NONE");
116:     break;

```

```
117:  default:
118:      break;
119:  }
120:      m_ItemInterfaceMessageID.IDToString(IDStr, sizeof(IDStr));
121:      printf("xTEDSItem name %s description %s interface %s type %s ID %s \n",m_strItemName,
m_strItemDescription, m_strItemInterfaceName, ItemTypeStr, IDStr);
122: }
123: #endif
```

File: sdm/common/xTEDS/SDMDataRates.h

```
1: #ifndef __SDM_DATA_RATES_H_
2: #define __SDM_DATA_RATES_H_
3:
4: enum SDMDataRates
5: {
6:     Event,
7:     Periodic
8: };
9:
10: #endif
```

File: sdm/common/xTEDS/xTEDSWrapperList.h

```
1: #ifndef _SDM_XTEDS_WRAPPER_LIST_H_
2: #define _SDM_XTEDS_WRAPPER_LIST_H_
3:
4: #include "MessageDef.h"
5: #include "xTEDSQualifierList.h"
6: #include "xTEDSWrapper.h"
7: #include "../message/SDMMessage_ID.h"
8:
9: class SDMLIB_API xTEDSWrapperListNode
10: {
11: public:
12: xTEDSWrapperListNode():Next(NULL),Data(NULL) {}
13: ~xTEDSWrapperListNode() {}
14: xTEDSWrapperListNode(const xTEDSWrapperListNode&);
15: xTEDSWrapperListNode& operator=(const xTEDSWrapperListNode&);
16:
17: xTEDSWrapperListNode* Next;
18: xTEDSWrapper* Data;
19: };
20:
21: class SDMLIB_API xTEDSWrapperList
22: {
23: public:
24: xTEDSWrapperList();
25: xTEDSWrapperList(const xTEDSWrapperList&);
26: xTEDSWrapperList& operator= (const xTEDSWrapperList&);
27: ~xTEDSWrapperList();
28:
29: void AddItem(xTEDSWrapper* Item);
30: MessageDef* RegInfo(const char* Name, const xTEDSQualifierList& Qualifiers, const char*
Interface) const;
31: MessageDef* RegexRegInfo(const char* Pattern, const xTEDSQualifierList& Qualifiers, const char*
Interface) const;
32: bool ContainsMessage(const SDMMessage_ID& RequestedId) const;
33: SDMMessage_ID GetFaultID(const SDMMessage_ID& MessageID) const;
34: SDMMessage_ID GetDataID(const SDMMessage_ID& MessageID) const;
35:
36: void PrintDebug() const;
37: private:
```

```
38: xTEDSWrapperListNode* Head;
39: void DeleteList();
40: };
41:
42: #endif
43:
```

File: sdm/common/Regex/RegexMatch.h

```
1: #ifndef _SDM_REGEX_MATCH_H_
2: #define _SDM_REGEX_MATCH_H_
3:
4: #include "../sdmLib.h"
5: #include "RegexCapture.h"
6:
7: class SDMLIB_API RegexMatch
8: {
9: public:
10: RegexMatch();
11: RegexMatch(int MaxCaptureSize);
12: RegexMatch(const RegexMatch& Right);
13: ~RegexMatch();
14: RegexMatch& operator=(const RegexMatch& Right);
15:
16: void SetMaxCaptureSize(int Size);
17: bool AddCapture(const RegexCapture& NewCapture);
18: const RegexCapture& operator[] (int index) const;
19: int FillMatchText (char* InputBuffer, int InputBufferSize) const;
20: void Clear();
21: private:
22: void DeleteCaptures();
23: RegexCapture* m_Captures;
24: int m_iNumCaptures;
25: int m_iCurCaptureIndex;
26: };
27:
28: #endif
29:
```

File: sdm/common/Regex/RegularExpression.cpp

```
1: #include "RegularExpression.h"
2:
3:
4: RegularExpression::RegularExpression() : m_CompiledExpression(), m_bExpressionSet(false)
5: {
6: }
7:
8: RegularExpression::~~RegularExpression()
9: {
10: FreeExpression();
11: }
12:
13: bool RegularExpression::Set(const char* strPattern)
14: {
15: FreeExpression();
16:
17: int iResult = regcomp(&m_CompiledExpression, strPattern, REG_EXTENDED);
18: if (iResult != 0)
19:     return false;
20:
21: m_bExpressionSet = true;
22: return true;
23: }
24:
25: void RegularExpression::FreeExpression()
26: {
27: if (m_bExpressionSet)
28: {
29:     regfree(&m_CompiledExpression);
30:     m_bExpressionSet = false;
31: }
32: }
```


File: sdm/common/Regex/Regex.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <sys/types.h>
4: #include <regex.h>
5: #include "../Exception/SDMRegexException.h"
6: #include "Regex.h"
7:
8: #ifdef __VXWORKS__
9: #include "pcreposix.h"
10: #endif
11:
12: RegexResult RegexSearchCapturesOnly(const char* SourceText, const RegularExpression& Pattern)
13: {
14:     regmatch_t Match[MAX_CAPTURES]; // Information for each match
15:     RegexResult TheResult;
16:     RegexCapture CurCapture;
17:     RegexMatch CurMatch(MAX_CAPTURES);
18:     int SourceOffset = 0;
19:     bool CaptureAdded = false;
20:     while (regexec(&Pattern.Get(), SourceText + SourceOffset, MAX_CAPTURES, Match, 0) == 0)
21:     {
22:         if (Match[0].rm_so == 0 && Match[0].rm_eo == 0)
23:             break;
24:         //
25:         // Index i starts at 1, which is the start of the captures, index 0 is the entire match
26:         CaptureAdded = false;
27:         for (int i = 1; i < MAX_CAPTURES; i++)
28:         {
29:             if (Match[i].rm_so == INVALID_CAPTURE)
30:                 break;
31:
32:             CurCapture.SetCaptureText(SourceText + SourceOffset + Match[i].rm_so, Match[i].rm_eo -
Match[i].rm_so);
33:             if (!CurMatch.AddCapture(CurCapture))
34:                 printf("%s - Error adding capture %d! \n", __FUNCTION__, i);
35:             else
36:                 CaptureAdded = true;
37:         }
38:         if (CaptureAdded)
```

```

39:     {
40:         TheResult.AddMatch(CurMatch);
41:         CurMatch.Clear();
42:     }
43:     // Match[0] is the matching for the entire regular expression
44:     // we want to skip ever looking at it again
45:     SourceOffset += Match[0].rm_eo;
46: }
47: return TheResult;
48: }
49:
50: bool DoesRegexMatch(const char* SourceText, const char* PatternText)
51: {
52:     regex_t CompiledExpression;
53:     if (0 != regcomp(&CompiledExpression, PatternText, REG_EXTENDED))
54:         throw SDMRegexException("Invalid regular expression specified. \n");
55:
56:     regmatch_t Match;
57:     bool Matched = false;
58:     if (regexec(&CompiledExpression, SourceText, 1, &Match, 0) == 0)
59:         Matched = true;
60:
61:     regfree(&CompiledExpression);
62:     return Matched;
63: }
64:
65: bool IsPatternValid(const char* PatternText)
66: {
67:     regex_t CompiledExpression;
68:     if (0 != regcomp(&CompiledExpression, PatternText, REG_EXTENDED))
69:         return false;
70:     regfree(&CompiledExpression);
71:     return true;
72: }

```

File: sdm/common/Regex/Makefile

```
1: include ../../Makefile.common
2: include ../../$(MAKEFILE_DEFS)
3:
4: .PHONY: all clean distclean
5:
6: BUILD_TARGETS=Regex RegexResult RegexMatch RegexCapture RegularExpression
7:
8: all: $(addsuffix .o,$(BUILD_TARGETS))
9:
10: %.o: %.cpp %.h
11: $(CXX) $(CXXFLAGS) -fPIC -c $<
12:
13: clean:
14: rm -f *.o
15:
16: distclean: clean
17: rm -f *~
```

File: sdm/common/Regex/RegexResult.cpp

```
1: #include "../MemoryUtils.h"
2: #include "RegexResult.h"
3: #include "../Exception/SDMBadIndexException.h"
4:
5: RegexResult::RegexResult() : m_Head(NULL), m_iNumMatches(0)
6: {
7: }
8:
9: RegexResult::~~RegexResult()
10: {
11: DeleteList();
12: }
13:
14: RegexResult::RegexResult(const RegexResult& Right) : m_Head(NULL),
m_iNumMatches(Right.m_iNumMatches)
15: {
16: DeleteList();
17:
18: MatchListNode* PrevNode = NULL, *CurLeft = NULL;
19: for (MatchListNode* CurRight = Right.m_Head; CurRight != NULL; CurRight = CurRight->Next)
20: {
21:     if (PrevNode == NULL)
22:     {
23:         CurLeft = m_Head = new MatchListNode();
24:         //if (CurLeft == NULL)
25:         //    SDMMemoryAllocError(__FUNCTION__);
26:     }
27:     else
28:     {
29:         CurLeft = new MatchListNode();
30:         //if (CurLeft == NULL)
31:         //    SDMMemoryAllocError(__FUNCTION__);
32:         PrevNode->Next = CurLeft;
33:     }
34:
35:     CurLeft->MatchData = CurRight->MatchData;
36:     PrevNode = CurLeft;
37: }
38: }
```

```

39:
40: RegexResult& RegexResult::operator=(const RegexResult& Right)
41: {
42: DeleteList();
43:
44: MatchListNode* PrevNode = NULL, *CurLeft = NULL;
45: for (MatchListNode* CurRight = Right.m_Head; CurRight != NULL; CurRight = CurRight->Next)
46: {
47:     if (PrevNode == NULL)
48:     {
49:         CurLeft = m_Head = new MatchListNode();
50:         //if (CurLeft == NULL)
51:         //    SDMMemoryAllocError(__FUNCTION__);
52:     }
53:     else
54:     {
55:         CurLeft = new MatchListNode();
56:         //if (CurLeft == NULL)
57:         //    SDMMemoryAllocError(__FUNCTION__);
58:         PrevNode->Next = CurLeft;
59:     }
60:
61:     CurLeft->MatchData = CurRight->MatchData;
62:     PrevNode = CurLeft;
63: }
64: this->m_iNumMatches = Right.m_iNumMatches;
65: return *this;
66: }
67:
68: void RegexResult::DeleteList()
69: {
70: for (MatchListNode* Cur = m_Head; Cur != NULL; )
71: {
72:     MatchListNode* Next = Cur->Next;
73:     delete Cur;
74:     Cur = Next;
75: }
76: m_Head = NULL;
77: }
78:
79: bool RegexResult::AddMatch(const RegexMatch& NewMatch)

```

```

80: {
81: if (m_Head == NULL)
82: {
83:     m_Head = new MatchListNode();
84:     //if (m_Head == NULL)
85:     //    SDMMemoryAllocError(__FUNCTION__);
86:
87:     m_Head->MatchData = NewMatch;
88: }
89: else
90: {
91:     // Find the insert position
92:     MatchListNode* Tail = NULL;
93:     for (Tail = m_Head; Tail->Next != NULL; Tail = Tail->Next)
94:         ;
95:
96:     Tail->Next = new MatchListNode();
97:     //if (Tail->Next == NULL)
98:     //    SDMMemoryAllocError(__FUNCTION__);
99:
100:     Tail->Next->MatchData = NewMatch;
101: }
102: m_iNumMatches++;
103: return true;
104: }
105:
106: const RegexMatch& RegexResult::operator[] (int Index) const
107: {
108:     int Count = 0;
109:     MatchListNode* Cur = NULL;
110:     for (Cur = m_Head; Cur != NULL; Cur = Cur->Next, Count++)
111:     {
112:         if (Count == Index)
113:             break;
114:     }
115:
116:     if (Cur == NULL)
117:         throw SDBadIndexException("Bad index in call to RegexResult::operator[]");
118:
119:     return Cur->MatchData;
120: }

```

File: sdm/common/Regex/RegexCapture.cpp

```
1: #include <string.h>
2: #include <stdlib.h>
3: #include "RegexCapture.h"
4: extern "C"
5: {
6: #include "../MemoryUtils.h"
7: }
8: #ifdef WIN32
9: # include "unistd.h"
10: #endif
11:
12: RegexCapture::RegexCapture() : m_strCaptureText(NULL)
13: {
14: }
15:
16: RegexCapture::~RegexCapture()
17: {
18: if (m_strCaptureText != NULL)
19:     free(m_strCaptureText);
20: }
21:
22: RegexCapture::RegexCapture(const RegexCapture& Right) : m_strCaptureText(NULL)
23: {
24: if (Right.m_strCaptureText != NULL)
25:     //this->m_strCaptureText = SDM_strdup(Right.m_strCaptureText);
26:     this->m_strCaptureText = strdup(Right.m_strCaptureText);
27: }
28:
29: RegexCapture& RegexCapture::operator= (const RegexCapture& Right)
30: {
31: if (this->m_strCaptureText != NULL)
32:     free(this->m_strCaptureText);
33: this->m_strCaptureText = NULL;
34:
35: if (Right.m_strCaptureText != NULL)
36:     //this->m_strCaptureText = SDM_strdup(Right.m_strCaptureText);
37:     this->m_strCaptureText = strdup(Right.m_strCaptureText);
38: return *this;
39: }
```

```
40:
41: bool RegexCapture::SetCaptureText(const char* NewCaptureText, int Length)
42: {
43: if (m_strCaptureText != NULL)
44:     free(m_strCaptureText);
45:
46: m_strCaptureText = SDM_strndup(NewCaptureText, Length);
47: return true;
48: }
49:
50: const char* RegexCapture::GetCaptureText() const
51: {
52: return m_strCaptureText;
53: }
54:
```


File: sdm/common/Regex/Regex.h

```
1: #ifndef _SDM_REGEX_H_
2: #define _SDM_REGEX_H_
3:
4: #include "../sdmLib.h"
5: #include "RegexResult.h"
6: #include "RegularExpression.h"
7:
8: #ifdef __VXWORKS__
9: #include "pcreposix.h"
10: #endif
11:
12: // The maximum number of captures to return
13: const int MAX_CAPTURES = 6;
14:
15: // Do not change the below value(s)
16: const int INVALID_CAPTURE = -1;
17:
18:
19: SDMLIB_API
20: RegexResult RegexSearchCapturesOnly(const char* SourceText, const RegularExpression& Pattern);
21: SDMLIB_API
22: bool DoesRegexMatch(const char* SourceText, const char* Pattern);
23: SDMLIB_API
24: bool IsPatternValid(const char* PatternText);
25:
26: #endif
```

File: sdm/common/Regex/RegexResult.h

```
1: #ifndef _SDM_REGEX_RESULT_H_
2: #define _SDM_REGEX_RESULT_H_
3:
4: #include "RegexMatch.h"
5: #include "../sdmLib.h"
6:
7: class SDMLIB_API MatchListNode
8: {
9: public:
10: MatchListNode() : MatchData(), Next(NULL) {}
11: MatchListNode(const MatchListNode& Right);
12: MatchListNode& operator=(const MatchListNode& Right);
13:
14: RegexMatch MatchData;
15: MatchListNode* Next;
16: };
17:
18: class SDMLIB_API RegexResult
19: {
20: public:
21: RegexResult();
22: ~RegexResult();
23: RegexResult(const RegexResult& Right);
24: RegexResult& operator=(const RegexResult& Right);
25:
26: bool AddMatch(const RegexMatch& NewMatch);
27:
28: int NumMatches() const { return m_iNumMatches; }
29: bool Matched() const { return m_iNumMatches != 0; }
30: const RegexMatch& operator[] (int Index) const;
31: private:
32: void DeleteList();
33: MatchListNode* m_Head;
34: int m_iNumMatches;
35: };
36:
37: #endif
38:
```

File: sdm/common/Regex/RegexMatch.cpp

```
1: #include "../MemoryUtils.h"
2: #include "RegexMatch.h"
3:
4: RegexMatch::RegexMatch() : m_Captures(NULL), m_iNumCaptures(0), m_iCurCaptureIndex(0)
5: {
6: }
7:
8: RegexMatch::RegexMatch(int MaxCaptureSize) : m_Captures(NULL), m_iNumCaptures(0),
m_iCurCaptureIndex(0)
9: {
10: SetMaxCaptureSize(MaxCaptureSize);
11: }
12:
13: RegexMatch::RegexMatch(const RegexMatch& Right) : m_Captures(NULL), m_iNumCaptures(0),
m_iCurCaptureIndex(0)
14: {
15: if (Right.m_iNumCaptures != 0)
16: {
17:     this->m_Captures = new RegexCapture[Right.m_iNumCaptures];
18:
19:     for (int i = 0; i < m_iNumCaptures; i++)
20:         this->m_Captures[i] = Right.m_Captures[i];
21:
22:     this->m_iCurCaptureIndex = Right.m_iCurCaptureIndex;
23: }
24: }
25:
26: RegexMatch::~~RegexMatch()
27: {
28: DeleteCaptures();
29: }
30:
31: RegexMatch& RegexMatch::operator= (const RegexMatch& Right)
32: {
33: DeleteCaptures();
34:
35: if (Right.m_iNumCaptures != 0)
36: {
37:     this->m_iNumCaptures = Right.m_iNumCaptures;
```

```

38:     this->m_Captures = new RegexCapture[Right.m_iNumCaptures];
39:
40:     for (int i = 0; i < m_iNumCaptures; i++)
41:         this->m_Captures[i] = Right.m_Captures[i];
42:
43:     this->m_iCurCaptureIndex = Right.m_iCurCaptureIndex;
44: }
45: return *this;
46: }
47:
48: void RegexMatch::DeleteCaptures()
49: {
50:     if (m_Captures != NULL)
51:     {
52:         delete [] m_Captures;
53:         m_Captures = NULL;
54:         m_iNumCaptures = 0;
55:         m_iCurCaptureIndex = 0;
56:     }
57: }
58:
59: void RegexMatch::SetMaxCaptureSize(int NewSize)
60: {
61:     DeleteCaptures();
62:
63:     m_Captures = new RegexCapture[NewSize];
64:
65:     m_iNumCaptures = NewSize;
66:     m_iCurCaptureIndex = 0;
67: }
68:
69: bool RegexMatch::AddCapture(const RegexCapture& NewCapture)
70: {
71:     if (m_Captures == NULL)
72:         return false;
73:     if (m_iCurCaptureIndex+1 == m_iNumCaptures || m_iCurCaptureIndex < 0)
74:         return false;
75:
76:     m_Captures[m_iCurCaptureIndex++] = NewCapture;
77:     return true;
78: }

```

```

79:
80: const RegexCapture& RegexMatch::operator[] (int Index) const
81: {
82:     return m_Captures[Index];
83: }
84:
85: int RegexMatch::FillMatchText(char* InputBuffer, int InputBufferSize) const
86: {
87:     int CurOffset = 0;
88:
89:     for (int i = 0; i < m_iCurCaptureIndex; i++)
90:     {
91:         strncpy(InputBuffer + CurOffset, m_Captures[i].GetCaptureText(), InputBufferSize - CurOffset);
92:         // +1 assures the null terminator is not overwritten
93:         CurOffset += strlen(InputBuffer + CurOffset) + 1;
94:     }
95:
96:     // Double null-terminate the last capture
97:     if (InputBufferSize - CurOffset >= 1)
98:         InputBuffer[++CurOffset] = '\0';
99:
100:     return CurOffset;
101: }
102:
103: void RegexMatch::Clear()
104: {
105:     m_iCurCaptureIndex = 0;
106: }
107:

```

File: sdm/common/Regex/RegexCapture.h

```
1: #ifndef _SDM_REGEX_CAPTURE_H_
2: #define _SDM_REGEX_CAPTURE_H_
3:
4: #include "../sdmLib.h"
5:
6: class SDMLIB_API RegexCapture
7: {
8: public:
9:     RegexCapture();
10:    ~RegexCapture();
11:    RegexCapture(const RegexCapture& Right);
12:    RegexCapture& operator=(const RegexCapture& Right);
13:
14:    bool SetCaptureText(const char* NewCaptureText, int Length);
15:    const char* GetCaptureText() const;
16: private:
17:    char* m_strCaptureText;
18: };
19:
20: #endif
21:
```

File: sdm/common/Regex/RegularExpression.h

```
1: #ifndef __SDM_REGULAR_EXPRESSION_H_
2: #define __SDM_REGULAR_EXPRESSION_H_
3:
4: #include <sys/types.h>
5: #ifdef __VXWORKS__
6: #include "pcreposix.h"
7: #else
8: #include <regex.h>
9: #endif
10:
11: #include "../message/SDMData.h"
12:
13: class SDMLIB_API RegularExpression
14: {
15: public:
16: RegularExpression();
17: ~RegularExpression();
18:
19: bool Set(const char* PatternText);
20:
21: const regex_t& Get() const { return m_CompiledExpression; }
22: private:
23: void FreeExpression();
24: regex_t m_CompiledExpression;
25:
26: bool m_bExpressionSet;
27: };
28:
29:
30: #endif
```

File: sdm/common/Time/TimeKeepingWin32.cpp

```
1: // TimeKeepingWin32.cpp : Sets up the shared memory for use by timeKeeping methods.
2: //
3:
4: #define WIN32_LEAN_AND_MEAN
5: #include <windows.h>
6: #include <stdio.h>
7:
8: #include "TimeKeeping.h"
9:
10: struct SDMLIB_API TimeKeepingRec* timeKeeping;
11: SDMLIB_API struct SDMLIB_API TimeKeepingRec* GetTimeKeeping(){ return timeKeeping; }
12:
13: static short deltaMonthDays[] = {0,1,-1,0,0,1,1,2,3,3,4,4,5 };
14: static unsigned int GpsSeconds()
15: {
16: // Leap seconds is set to 14, which was last updated on 31 Dec, 2005. As of this date, Nov 15, 2007,
the next
17: // update is unannounced.
18: unsigned int seconds;
19: unsigned int leapSeconds = 14;
20: SYSTEMTIME systemTime;
21: GetSystemTime( &systemTime );
22: short yearNow = systemTime.wYear;
23: short dYear = systemTime.wYear - 1980;
24: short dMonth = systemTime.wMonth - 1; // January
25: short dDay = systemTime.wDay - 6;
26: short feb29=0;
27: int thisYearLeaps;
28: for( short year = 1980; year<= yearNow; year++ )
29:     if ( !(year%4) && ( (year%100) || !(year%400) ) )
30:     {
31:         thisYearLeaps = true;
32:         feb29++;
33:     }
34:     else
35:         thisYearLeaps = false;
36: if ( thisYearLeaps && dMonth < 2 ) feb29--;
37: unsigned int days = dYear*365 + dMonth*30 + deltaMonthDays[dMonth] + dDay + feb29;
```



```

38: seconds    =    days*86400    +    (systemTime.wHour*60    +    systemTime.wMinute)*60    +
systemTime.wSecond + leapSeconds;
39:
40: return seconds;
41: }
42:
43: SDMLIB_API void InitializeTimeKeeping()
44: {
45: #define BUF_SIZE 256
46: TCHAR szName[]=TEXT("TimeKeeping");
47:
48: HANDLE hMapFile = OpenFileMapping(
49:     FILE_MAP_ALL_ACCESS, // read/write access
50:     FALSE,                // do not inherit the name
51:     szName);              // name of mapping object
52:
53: bool created = false;
54:
55: if (hMapFile == NULL)
56: {
57: //      printf("Couldn't OpenFileMapping; trying to CreateFileMapping \n");
58:
59: hMapFile = CreateFileMapping(
60:     INVALID_HANDLE_VALUE, // use paging file
61:     NULL,                 // default security
62:     PAGE_READWRITE,       // read/write access
63:     0,                    // max. object size
64:     BUF_SIZE,             // buffer size
65:     szName);              // name of mapping object
66:
67: if (hMapFile == NULL || hMapFile == INVALID_HANDLE_VALUE)
68: {
69:     printf("Could not create file mapping object (%d). \n",
70:         GetLastError());
71:     return;
72: }
73:     created = true;
74: }
75:
76: timeKeeping = (struct SDMLIB_API TimeKeepingRec*) MapViewOfFile(hMapFile, // handle to
map object

```

```

77:    FILE_MAP_ALL_ACCESS, // read/write permission
78:    0,
79:    0,
80:    BUF_SIZE );
81:
82: if (timeKeeping == NULL)
83: {
84:     printf("Could not map view of file (%d) \n",
85:         GetLastError());
86:     return;
87: }
88:
89: if ( created )
90: {
91:     timeKeeping->tareCount = 0;
92:     timeKeeping->mode=0; // 0=unsynched 1=unscaled, free running; 2=time scaled, free running;
93:         // 3 = unscaled, paused, 4 = scaled, paused
94:     timeKeeping->gpsEpochTareSeconds=0;
95:     timeKeeping->ratio=1.0;
96:     timeKeeping->uSecScale=1.0;
97:     timeKeeping->nSecScale=1.0;
98:     timeKeeping->perfCountFreq=1000000000;
99:
100:     LARGE_INTEGER perfCountFreq;
101:     QueryPerformanceFrequency( &perfCountFreq );
102:     timeKeeping->perfCountFreq = perfCountFreq.QuadPart;
103:     timeKeeping->uSecScale = 1000000.0 / (double)timeKeeping->perfCountFreq;
104:     // For future use, just to show that QueryPerformanceCounter resolves down to <1 ns
105:     timeKeeping->nSecScale = 1000000000.0 / (double)timeKeeping->perfCountFreq;
106:     LARGE_INTEGER count;
107:     QueryPerformanceCounter( &count );
108:     timeKeeping->tareCount = count.QuadPart;
109:     timeKeeping->gpsEpochTareSeconds = GpsSeconds();
110: }
111: }

```

File: sdm/common/Time/TimeKeepingLinux.cpp

```
1: // TimeKeepingLinux.cpp : Sets up the shared memory for use by timeKeeping methods.
2: //
3:
4: #include <stdio.h>
5: #include <stdlib.h>
6: #include <sys/types.h>
7: #include <sys/stat.h>
8: #ifndef __VXWORKS__
9: #include <sys/ipc.h>
10: #include <sys/shm.h>
11: #else
12: #include <sys/mman.h>
13: #include <unistd.h>
14: #include <fcntl.h>
15: #endif
16: #include <sys/time.h>
17: #include <time.h>
18: #include <errno.h>
19: #include <string.h>
20:
21: #include "TimeKeeping.h"
22:
23: struct SDMLIB_API TimeKeepingRec* timeKeeping;
24: SDMLIB_API struct SDMLIB_API TimeKeepingRec* GetTimeKeeping(){ return timeKeeping; }
25:
26: static bool g_bTimeInitialized = false;
27: bool IsTimeInitialized() { return g_bTimeInitialized; }
28:
29: static unsigned int GpsSeconds()
30: {
31: // Leap seconds is set to 14, which was last updated on 31 Dec, 2005. As of this date, Nov 15, 2007,
the next
32: // update is unannounced.
33: unsigned int seconds;
34: unsigned int leapSeconds = 14;
35:
36: struct tm gpsEpoch;
37: memset(&gpsEpoch,0,sizeof(gpsEpoch));
38: gpsEpoch.tm_year = 80;
```

```

39: gpsEpoch.tm_mday = 6;
40: time_t gpsTime = mktime(&gpsEpoch);
41: time_t now;
42: time(&now);
43: seconds = now - gpsTime + leapSeconds;
44:
45: return seconds;
46: }
47:
48: SDMLIB_API void InitializeTimeKeeping()
49: {
50: #define SHM_KEY 0x43258902
51: #define SHM_SIZE 256
52:
53: void* pAddr;
54: bool bInitialize = false;
55: int  shmid;
56:
57:
58: #ifndef __VXWORKS__
59: key_t key = (key_t)SHM_KEY;
60:
61: if((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT | IPC_EXCL)) == -1)
62: {
63:     if((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1)
64: {
65:     perror("shmget");
66:     return;
67: }
68:
69:     bInitialize = false;
70: }
71: else
72: {
73:     bInitialize = true;
74: }
75:
76: if((pAddr=shmat(shmid, (void*)0, 0)) == (void*)-1)
77: {
78:     perror("shmat");
79:     return;

```

```

80: }
81: #else
82: if((shmid = shm_open("/timeKeepingShm", O_CREAT | O_RDWR , S_IRUSR | S_IWUSR)) < 0)
83: {
84:     if((shmid = shm_open("/timeKeepingShm", O_RDWR, S_IRUSR | S_IWUSR)) < 0)
85:     {
86:         printf("Error creating shared memory for time keeping \n");
87:         return;
88:     }
89:     bInitialize = false;
90: }
91: else
92: {
93:     bInitialize = true;
94: }
95:
96: //set object size
97:
98: if (ftruncate(shmid, 0x40) == -1)
99: {
100:     printf("Error setting shm size \n");
101:     return;
102: }
103: pAddr = mmap(0, 0x40, PROT_READ | PROT_WRITE, MAP_SHARED, shmid, 0);
104: if (pAddr == (void*)MAP_FAILED)
105: {
106:     printf("Error mapping shm \n");
107:     return;
108: }
109: close(shmid);
110: #endif
111: timeKeeping = (struct SDMLIB_API TimeKeepingRec*)pAddr;
112:
113: if(bInitialize)
114: {
115:     timeKeeping->tareCount = 0;
116:     timeKeeping->mode=0; // 0=unsynced 1=unscaled, free running; 2=time scaled, free running;
117:     // 3 = unscaled, paused, 4 = scaled, paused
118:     timeKeeping->gpsEpochTareSeconds=0;
119:     timeKeeping->ratio=1.0;
120:     timeKeeping->uSecScale=1.0;

```

```

121:    timeKeeping->nSecScale=1.0;
122:    timeKeeping->perfCountFreq=1000000000;
123:
124: #ifdef __VXWORKS__
125:    struct timespec tv;
126:    clock_gettime(CLOCK_REALTIME, &tv);
127:    timeKeeping->tareCount = tv.tv_sec * 1000000LL + tv.tv_nsec/1000;
128: #else
129:    struct timeval tv;
130:    gettimeofday(&tv, NULL);
131:    timeKeeping->tareCount = tv.tv_sec * 1000000LL + tv.tv_usec;
132: #endif
133:    timeKeeping->perfCountFreq = 1000000;
134:    timeKeeping->uSecScale = 1.0;
135:    timeKeeping->nSecScale = 1000.0;
136:
137:    timeKeeping->gpsEpochTareSeconds = GpsSeconds();
138: }
139: g_bTimeInitialized = true;
140: }

```

File: sdm/common/Time/SDMTime.cpp

```
1:
2:
3:
4:
5:
6: //-----
7: //-----RTEMS IMPLEMENTATION-----
8: //-----
9: #elif defined (RTEMS_BUILD)
10: /*
11:  * Gets the current system time and stores the current seconds in SecondsOut, and the current
12:  * microseconds value in USecondsOut.
13:  * Params:
14:  *     SecondsOut [OUTPUT] - Pointer location to store the current number of seconds
15:  *     USecondsOut [OUTPUT] - Pointer location to store the current number of micro seconds
16:  */
17: void SDM_GetTime(unsigned int *SecondsOut, unsigned int *USecondsOut)
18: {
19: if (SecondsOut == NULL || USecondsOut == NULL) return;
20: #ifdef WIN32
21: //Get and return the time from Windows
22: time_t CurSeconds;
23: time(&CurSeconds);
24: *SecondsOut = static_cast<long>(SecondsOut);
25:
26: SYSTEMTIME SysTime;
27: GetSystemTime(&SysTime);
28: *USecondsOut = static_cast<long>(SysTime.wMilliseconds);
29: #elif defined (RTEMS_BUILD)
30: //Get and return the time from RTEMS
31: #else
32: //Get and return the time from Linux
33: struct timeval CurrTime;
34: gettimeofday(&CurrTime, NULL);
35: *SecondsOut = static_cast<long>(CurrTime.tv_sec);
36: *USecondsOut = static_cast<long>(CurrTime.tv_usec);
37: #endif
38: }
39: /*
```

```

40: * Sleep for the time specified in Seconds and USeconds.
41: * Params:
42: *     Seconds - The number of seconds to sleep
43: *     USeconds - The number of microseconds to sleep
44: *
45: */
46: void SDM_Sleep(unsigned int Seconds, unsigned int USeconds)
47: {
48:     sleep(Seconds);
49:     usleep(USeconds);
50: }
51: /*
52: * Create a timer to expire at the specified number of Seconds and USeconds. This function returns
53: * an ID number for the timer to distinguish between multiple timers, and to perform other timer
54: * related operations on the ID. Optionally, the function specified by Callback will be called
55: * when the timer expires.
56: * Params:
57: *     Seconds - The number of seconds until the timer is to expire
58: *     USeconds - The number of microseconds until the timer is to expire
59: *     Callback - The function to call when the timer expires (can be NULL).
60: *     Periodic - Whether the timer should auto-restart upon expiration.
61: * Returns:
62: *     int - An identifier number for the timer.
63: */
64: int SDM_CreateTimer(unsigned int Seconds, unsigned int USeconds, void(*Callback)(int), bool
Periodic)
65: {
66:
67: }
68: /*
69: * Deletes the timer specified by TimerID. This function has no effect if the timer specified does
70: * not exist.
71: *
72: */
73: void SDM_DeleteTimer(int TimerID)
74: {
75:
76: }
77: /*
78: * Reads the current value for the TimerID specified. The time returned in the output
79: * parameters is the time remaining until the timer expires.

```



```

80: *   Params:
81: *       TimerID - Identifier number of the timer
82: *       SecondsOut [OUTPUT] - Pointer to the location where to store the seconds remaining
83: *       USecondsOut [OUTPUT] - Pointer to the location where to store the micro seconds
remaining
84: *   Returns:
85: *       int - 0 on success, or -1 if some error occurred
86: *
87: */
88: int SDM_ReadTimer(int TimerID, unsigned int *SecondsOut, unsigned int *USecondsOut);
89: {
90:
91: }
92:
93:
94:
95:
96: #endif

```

File: sdm/common/Time/LookupTable.h

```
1: #ifndef _LOOKUP_TABLE_H_
2: #define _LOOKUP_TABLE_H_
3: #include "../sdmLib.h"
4: #include <pthread.h>
5:
6: #define MAX_TABLE_ITEMS 5
7: #define KEY_EMPTY -1
8:
9: struct Item
10: {
11:     int Key;
12:     unsigned int Value;
13:     pthread_t ThreadID;
14:     Item(): Key(KEY_EMPTY), Value(0), ThreadID() {}
15: };
16:
17: class SDMLIB_API LookupTable
18: {
19: public:
20:     LookupTable();
21:     unsigned int GetValueFromKey(int Key);
22:     int GetKeyFromValue(unsigned int Value);
23:     int AddValue(unsigned int Value);
24:     unsigned int RemoveKeyGetValue(int Key);
25:     int RemoveValueGetKey(unsigned int Value);
26:     void UpdateValue(int Key, unsigned int NewValue);
27:     bool UpdateThreadIDFromKey(int Key, pthread_t ThreadID);
28:     bool UpdateThreadIDFromValue(unsigned int Value, pthread_t ThreadID);
29:     bool GetThreadIDFromKey(int Key, pthread_t *ThreadIDOut);
30: private:
31:     Item TableItems[MAX_TABLE_ITEMS];
32:     int CurKey;
33: };
34:
35: #endif
```

File: sdm/common/Time/TimerList.cpp

```
1: #include <unistd.h>
2: #include <stdlib.h>
3: #include "TimerList.h"
4:
5: #include <stdio.h>
6:
7: TimerList::TimerList():CurID(1),CurRunningTimer(-1)
8: {
9: }
10: /*
11: *
12: *
13: */
14: int TimerList::AddTimer(unsigned int Seconds, unsigned int USeconds, unsigned int CurSeconds,
unsigned int CurUSeconds, void *Callback, bool Periodic, unsigned int AssignedID)
15: {
16: //Find an inactive timer entry
17: int i = 0;
18: for (i = 0; i < MAX_NUM_TIMERS; i++)
19:     if (Timers[i].TimerID == 0)
20:         break;
21: //If no entries, return error
22: if (i == MAX_NUM_TIMERS) return -1;
23: //i is the index of an available timer entry
24: Timers[i].TimerVal.SetSeconds(Seconds);
25: Timers[i].TimerVal.SetUSeconds(USeconds);
26: Timers[i].StartTime.SetSeconds(CurSeconds);
27: Timers[i].StartTime.SetUSeconds(CurUSeconds);
28: Timers[i].Callback = Callback;
29: Timers[i].TimerID = AssignedID;
30: Timers[i].Periodic = Periodic;
31: return Timers[i].TimerID;
32: }
33: /*
34: *
35: *
36: */
37: int TimerList::AddTimer(unsigned int Seconds, unsigned int USeconds, unsigned int CurSeconds,
unsigned int CurUSeconds, void *Callback, bool Periodic)
```

```

38: {
39: //Find an inactive timer entry
40: int i = 0;
41: for (i = 0; i < MAX_NUM_TIMERS; i++)
42:     if (Timers[i].TimerID == 0)
43:         break;
44: //If no entries, return error
45: if (i == MAX_NUM_TIMERS) return -1;
46: //i is the index of an available timer entry
47: Timers[i].TimerVal.SetSeconds(Seconds);
48: Timers[i].TimerVal.SetUSeconds(USeconds);
49: Timers[i].StartTime.SetSeconds(CurSeconds);
50: Timers[i].StartTime.SetUSeconds(CurUSeconds);
51: Timers[i].Callback = Callback;
52: Timers[i].TimerID = CurID++;
53: Timers[i].Periodic = Periodic;
54: return Timers[i].TimerID;
55: }
56: /*
57:  * Return the timer id of the timer that most recently expired
58:  *
59:  */
60: int TimerList::WhichExpired(unsigned int CurSeconds, unsigned int CurUSeconds)
61: {
62: //We have the current time, find the timer entry whose difference in time is the smallest
63: SecTime CurTime(CurSeconds, CurUSeconds);
64: SecTime SmallestTime(0x7FFFFFFF, 0x7FFFFFFF);
65: int CurIndex = -1;
66: int i = 0;
67: //Find the entry with the minimal time difference
68: for (i = 0; i < MAX_NUM_TIMERS; i++)
69: {
70:     if (Timers[i].TimerID == TIMER_INACTIVE) continue;
71:     //See if this timer's start time plus its timer value is less than the current time
72:     SecTime TimerExpireTime = Timers[i].TimerVal + Timers[i].StartTime;
73:     if (TimerExpireTime < CurTime)
74:     {
75:         if (TimerExpireTime < SmallestTime)
76:         {
77:             SmallestTime = TimerExpireTime;
78:             CurIndex = i;

```

```

79:     }
80: }
81: }
82: //If we could not find a timer
83: if (CurIndex == -1) return -1;
84: //Otherwise, i is the timer found
85: //Save the new current time for the timer
86: Timers[CurIndex].StartTime = CurTime;
87: return Timers[CurIndex].TimerID;
88: }
89: /*
90:  * Remove a timer from the list.
91:  *
92:  */
93: bool TimerList::RemoveTimer(int TimerID)
94: {
95: //Find the timer
96: int TimerIndex = 0;
97: if ((TimerIndex = GetIndexOfID(TimerID)) == TIMER_NOT_FOUND)
98:     return false;
99: //Delete item, set the entry as inactive
100: Timers[TimerIndex].TimerVal.Clear();
101: Timers[TimerIndex].Callback = NULL;
102: Timers[TimerIndex].TimerID = 0; //Inactive flag
103: return true;
104: }
105: /*
106:  * Get the callback pointer for the TimerID.
107:  *
108:  */
109: void* TimerList::GetCallback(int TimerID)
110: {
111: //Find the timer
112: int TimerIndex = 0;
113: if ((TimerIndex = GetIndexOfID(TimerID)) == TIMER_NOT_FOUND)
114:     return NULL;
115: return Timers[TimerIndex].Callback;
116: }
117: /*
118:  * Get whether the timer is periodic.
119:  *

```

```

120: */
121: bool TimerList::IsPeriodic(int TimerID)
122: {
123:     //Find the timer
124:     int TimerIndex = 0;
125:     if ((TimerIndex = GetIndexOfID(TimerID)) == TIMER_NOT_FOUND)
126:         return false;
127:     return Timers[TimerIndex].Periodic;
128: }
129: /*
130: * The the smallest interval to the next timer expiration, returns the ID of that timer.
131: *
132: */
133: int TimerList::GetSmallestTimerInterval(unsigned int CurSeconds, unsigned int CurUSeconds,
unsigned int *TimerSecOut, unsigned int *TimerUsecOut)
134: {
135:     if (TimerSecOut == NULL || TimerUsecOut == NULL) return -1;
136:     //We have the current time, find the timer entry whose difference in time is the smallest
137:     const SecTime CurTime(CurSeconds, CurUSeconds);
138:     const SecTime TimeZero(0,0);
139:     SecTime SmallestTime(0x7FFFFFFF, 0x7FFFFFFF);
140:     SecTime CurLowPriTime(-1000000, -1000000);
141:     int CurIndex = -1;
142:     int i = 0;
143:     //Find the entry with the minimal time difference
144:     for (i = 0; i < MAX_NUM_TIMERS; i++)
145:     {
146:         if (Timers[i].TimerID == TIMER_INACTIVE) continue;
147:         //Find the timer whose start time plus its timer time is the smallest
148:         SecTime TimerExpireTime = Timers[i].TimerVal + Timers[i].StartTime;
149:         if (TimerExpireTime - CurTime < TimeZero)
150:             continue;
151:         else
152:         {
153:             if (TimerExpireTime - CurTime < SmallestTime)
154:             {
155:                 SmallestTime = TimerExpireTime - CurTime;
156:                 CurIndex = i;
157:             }
158:         }
159:     }

```

```

160:
161: //If we could not find a timer
162: if (CurIndex == -1)
163: {
164:     *TimerSecOut = *TimerUsecOut = 0;
165:     return -1;
166: }
167:
168: *TimerSecOut = (unsigned int)abs(SmallestTime.GetSeconds());
169: *TimerUsecOut = (unsigned int)abs(SmallestTime.GetUSeconds());
170:
171: if (CurIndex >= 0 && CurIndex < MAX_NUM_TIMERS)
172: {
173:     return Timers[CurIndex].TimerID;
174: }
175:
176: return -1;
177: }
178: /*
179:  * Get the time remaining on a timer.
180:  *
181:  */
182: int TimerList::GetTimeRemaining(int TimerID, unsigned int CurSeconds, unsigned int
CurUSeconds, unsigned int *TimerSecOut, unsigned int *TimerUsecOut)
183: {
184:     if (TimerSecOut == NULL || TimerUsecOut == NULL) return -1;
185:     //Find the timer
186:     int TimerIndex = 0;
187:     if ((TimerIndex = GetIndexOfID(TimerID)) == TIMER_NOT_FOUND)
188:         return -1;
189:
190:     SecTime TimeRemaining(CurSeconds, CurUSeconds);
191:     TimeRemaining = Timers[TimerIndex].TimerVal - (TimeRemaining -
Timers[TimerIndex].StartTime);
192:     *TimerSecOut = TimeRemaining.GetSeconds();
193:     *TimerUsecOut = TimeRemaining.GetUSeconds();
194:     return 0;
195: }
196: /*
197:  * Reset a timer's start time.
198:  *

```

```

199: */
200: void TimerList::TimerFired(int TimerID, unsigned int Seconds, unsigned int USeconds)
201: {
202:     //Find the timer
203:     int TimerIndex = 0;
204:     if ((TimerIndex = GetIndexOfID(TimerID)) == TIMER_NOT_FOUND)
205:         return;
206:
207:     SecTime NewTime(Seconds, USeconds);
208:     Timers[TimerIndex].StartTime = NewTime;
209: }
210: /*
211:  * Get the index of a TimerID.
212:  *
213:  */
214: int TimerList::GetIndexOfID(int ID)
215: {
216:     for (int i = 0; i < MAX_NUM_TIMERS; i++)
217:         if (Timers[i].TimerID == ID)
218:             return i;
219:     return -1;
220: }
221: /*
222:  * Get whether the list is empty.
223:  *
224:  */
225: bool TimerList::IsEmpty()
226: {
227:     for (int i = 0; i < MAX_NUM_TIMERS; i++)
228:     {
229:         if (Timers[i].TimerID != TIMER_INACTIVE)
230:             return false;
231:     }
232:     return true;
233: }
234: /*
235:  * Set the current running timer
236:  *
237:  */
238: void TimerList::SetRunningTimer(int TimerID)
239: {

```



```
240:   if (TimerID >=0 && TimerID < CurID)
241:       CurRunningTimer = TimerID;
242: }
```

File: sdm/common/Time/SDMTimeLinux.cpp

```
1: //-----
2: //-----LINUX IMPLEMENTATION-----
3: //-----
4: #include <sys/time.h>
5: #include <unistd.h>
6: #include <signal.h>
7: #include <pthread.h>
8: #include <stdio.h>
9: #include "TimerList.h"
10: #include "SDMTimeLinux.h"
11: #include "TimeKeeping.h"
12:
13: #ifdef __VXWORKS__
14: #include <vxWorks.h>
15: #include <time.h>
16: #include <timers.h >
17: #endif
18:
19: #define DEBUG_TIMER    1
20:
21: static TimerList Timers;
22: static pthread_mutex_t TimerMutex = PTHREAD_MUTEX_INITIALIZER;
23: static bool HandlerThreadStarted = false;
24: static pthread_mutex_t HandlerStartedMutex = PTHREAD_MUTEX_INITIALIZER;
25:
26: /*
27:  * Initializes the timer service. It prevents user program threads from being able to handle
28:  * SIGALRM. This problem is only applicable to the Linux and RTEMS builds.
29:  *
30:  *
31:  */
32: void SDM_TimeInit()
33: {
34: //Set the global thread mask to ignore SIGALRM
35: sigset_t SignalMask;
36: sigemptyset(&SignalMask);
37: sigaddset(&SignalMask, SIGALRM);
38: if ((pthread_sigmask(SIG_BLOCK, &SignalMask, NULL)) != 0)
39:     perror("SDM_TimeInit::pthread_sigmask(): ");
```

```

40: }
41: /*
42: * Gets the current system time and stores the current seconds in SecondsOut, and the current
43: * microseconds value in USecondsOut.
44: * Params:
45: *     SecondsOut [OUTPUT] - Pointer location to store the current number of seconds
46: *     USecondsOut [OUTPUT] - Pointer location to store the current number of micro seconds
47: */
48:
49: /* If this is Linux or uClinux with a FSWIL time source */
50: #if !defined(__uClinux__) || defined(BUILD_FSWIL_TIMESYNC_TIME_SOURCE)
51: void SDM_GetTime(unsigned int *SecondsOut, unsigned int *USecondsOut)
52: {
53: if (SecondsOut == NULL || USecondsOut == NULL) return;
54: if (false == IsTimeInitialized())
55: {
56:     InitializeTimeKeeping();
57: }
58:
59: #ifdef __VXWORKS__ //VxWorks has nanosecond precision
60: struct timespec tv;
61: int tv_usec;
62: clock_gettime(CLOCK_REALTIME, &tv);
63: tv_usec = tv.tv_nsec/1000;
64: long long scaledCount = tv.tv_sec*1000000LL - timeKeeping->tareCount + tv_usec;
65:
66: if ( timeKeeping->mode == 1 )
67: {
68:     scaledCount = (long long)(scaledCount*timeKeeping->ratio);
69: }
70: unsigned long sec = (unsigned long)(scaledCount / timeKeeping->perfCountFreq);
71: long long seconds = sec * timeKeeping->perfCountFreq;
72: *SecondsOut = (unsigned int)(sec + timeKeeping->gpsEpochTareSeconds);
73: *USecondsOut = (unsigned int)((scaledCount-seconds) * timeKeeping->uSecScale);
74: #else
75: struct timeval tv;
76: gettimeofday(&tv, NULL);
77: long long scaledCount = tv.tv_sec*1000000LL - timeKeeping->tareCount + tv.tv_usec;
78: if ( timeKeeping->mode == 1 )
79: {
80:     scaledCount = (long long)(scaledCount*timeKeeping->ratio);

```

```

81: }
82: unsigned long sec = (unsigned long)(scaledCount / timeKeeping->perfCountFreq);
83: long long seconds = sec * timeKeeping->perfCountFreq;
84: *SecondsOut = (unsigned int)(sec + timeKeeping->gpsEpochTareSeconds);
85: *USecondsOut = (unsigned int)((scaledCount-seconds) * timeKeeping->uSecScale);
86: #endif
87: }
88: #else /* Otherwise, build for uClinux with a low-level TAT message */
89:
90: struct sysclock {
91:     unsigned long CLOCK_VER;
92:     unsigned long CLOCK_STAT;
93:     unsigned long CLOCK_CTRL;
94:     unsigned long TIME_SEC;
95:     unsigned long TIME_SUBSEC;
96:     unsigned long TAT_SEC;
97:     unsigned long TAT_SUBSEC;
98:     unsigned long RES_SUBSEC;
99: } __attribute__((packed));
100:
101: #warning "Unmaintainable definition"
102: #define CONFIG_XILINX_SYSCLOCK_PNP_0_BASEADDR 0x81610000
103:
104: void SDM_GetTime(unsigned int *SecondsOut, unsigned int *USecondsOut)
105: {
106:     if (SecondsOut == NULL || USecondsOut == NULL) return;
107:     struct timeval tv;
108:     volatile struct sysclock *ssysclk = (struct
sysclock*)(CONFIG_XILINX_SYSCLOCK_PNP_0_BASEADDR);
109:
110:     unsigned long sec1;
111:     unsigned long sec2;
112:     unsigned long subs;
113:
114:     sec1 = ssysclk->TIME_SEC;
115:
116:     if (sec1 != 0) {
117:         subs = ssysclk->TIME_SUBSEC;
118:         sec2 = ssysclk->TIME_SEC;
119:
120:         /*

```

```

121:     * Detect rollover. We have one second
122:     * to win the race.
123:     */
124:     if (sec1 != sec2) {
125:         sec1 = ssysclk->TIME_SEC;
126:         subs = ssysclk->TIME_SUBSEC;
127:     }
128:
129:     *SecondsOut = sec1;
130:     *USecondsOut = subs;
131: }
132: else {
133:     /*
134:     * Clock not running?
135:     */
136:     gettimeofday(&tv, NULL);
137:
138:     *SecondsOut = tv.tv_sec;
139:     *USecondsOut = tv.tv_usec;
140: }
141: }
142: #endif
143:
144: /*
145: * Gets the current system time and returns it as a SecTime
146: * Params:
147: *     SecTime [INPUT/OUTPUT] - Reference to a SecTime structure
148: */
149: void SDM_GetTime( SecTime &secTime )
150: {
151:     unsigned int uiSec;
152:     unsigned int uiUsec;
153:
154:     SDM_GetTime(&uiSec, &uiUsec);
155:     secTime.SetSeconds(uiSec);
156:     secTime.SetUSeconds(uiUsec);
157: }
158:
159: /*
160: * Sleep for the time specified in Seconds and USeconds.
161: * Params:

```

```

162: *      Seconds - The number of seconds to sleep
163: *      USeconds - The number of microseconds to sleep
164: *
165: */
166: void SDM_Sleep(unsigned int Seconds, unsigned int USeconds)
167: {
168: #ifndef __uCLinux__
169:   unsigned int secs = (unsigned int)(Seconds / timeKeeping->ratio);
170:   unsigned int usecs = (unsigned int)(USeconds / timeKeeping->ratio);
171:
172:   sleep(secs);
173:   usleep(usecs);
174: #else
175:   sleep(Seconds);
176:   usleep(USeconds);
177: #endif
178: }
179: /*
180: * Create a timer to expire at the specified number of Seconds and USeconds. This function returns
181: * an ID number for the timer to distinguish between multiple timers, and to perform other timer
182: * related operations on the ID. Optionally, the function specified by Callback will be called
183: * when the timer expires.
184: * Params:
185: *      Seconds - The number of seconds until the timer is to expire
186: *      USeconds - The number of microseconds until the timer is to expire
187: *      Callback - The function to call when the timer expires (can be NULL).
188: *      Periodic - Whether the timer should auto-restart upon expiration.
189: * Returns:
190: *      int - An identifier number for the timer.
191: */
192: #ifndef __VXWORKS__ //For all version except VxWorks
193: int SDM_CreateTimer(unsigned int Seconds, unsigned int USeconds, void(*Callback)(int), bool
Periodic)
194: {
195:   //Start the signal handling thread
196:   pthread_mutex_lock(&HandlerStartedMutex);
197:   // If the handler needs to be started
198:   if (!HandlerThreadStarted)
199:   {
200:     //Set the global thread mask to ignore SIGALRM
201:     sigset_t SignalMask;

```

```

202:     sigemptyset(&SignalMask);
203:     sigaddset(&SignalMask, SIGALRM);
204:     if ((pthread_sigmask(SIG_BLOCK, &SignalMask, NULL)) != 0)
205:     {
206:         perror("SDM_CreateTimer::pthread_sigmask(): ");
207:         pthread_mutex_unlock(&HandlerStartedMutex);
208:         return -1;
209:     }
210:     //Start the HandlerThread
211:     pthread_t HandlerThread;
212:     if (pthread_create (&HandlerThread, NULL, TimerHandler, NULL) != 0)
213:     {
214:         perror("SDM_CreateTimer::pthread_create(): ");
215:         pthread_mutex_unlock(&HandlerStartedMutex);
216:         return -1;
217:     }
218:     pthread_detach(HandlerThread);
219:     //Now the thread has started, update the HandlerThreadStarted
220:     HandlerThreadStarted = true;
221: }
222: pthread_mutex_unlock(&HandlerStartedMutex);
223:
224: //Get the current time
225: unsigned int CurSec, CurUsec;
226: SDM_GetTime(&CurSec, &CurUsec);
227: //Add the timer to the list
228: pthread_mutex_lock(&TimerMutex);
229:
230: int TimerNum = Timers.AddTimer(Seconds, USeconds, CurSec, CurUsec, (void*)Callback,
Periodic);
231: bool TimerListEmpty = Timers.IsEmpty();
232: //See if there is already a timer running
233:
234: itimerval TimeVal;
235: getitimer(ITIMER_REAL, &TimeVal);
236:
237: if (TimerListEmpty || (TimeVal.it_value.tv_sec == 0 && TimeVal.it_value.tv_usec == 0))
238: {
239:     Timers.SetRunningTimer(TimerNum);
240:     //Start a new timer
241:     TimeVal.it_value.tv_sec = Seconds;

```

```

242:     TimeVal.it_value.tv_usec = USeconds;
243:
244:     setitimer(ITIMER_REAL, &TimeVal, NULL);
245:
246: }
247: //If the current running timer won't expire before this one needs to, reset it to this timer
248: else if (static_cast<unsigned int>(TimeVal.it_value.tv_sec) > Seconds || (static_cast<unsigned
int>(TimeVal.it_value.tv_sec) == Seconds && static_cast<unsigned int>(TimeVal.it_value.tv_usec) >
USeconds))
249: {
250:     Timers.SetRunningTimer(TimerNum);
251:     TimeVal.it_value.tv_sec = Seconds;
252:     TimeVal.it_value.tv_usec = USeconds;
253:
254:     setitimer(ITIMER_REAL, &TimeVal, NULL);
255: }
256: pthread_mutex_unlock(&TimerMutex);
257: return TimerNum;
258: }
259:
260:
261: /*
262: * Deletes the timer specified by TimerID. This function has no effect if the timer specified does
263: * not exist.
264: *
265: */
266: void SDM_DeleteTimer(int TimerID)
267: {
268:     //Get the current time
269:     unsigned int CurSec, CurUsec, t1, t2;
270:     SDM_GetTime(&CurSec, &CurUsec);
271:
272:     pthread_mutex_lock(&TimerMutex);
273:     int TempID = Timers.GetRunningTimer();
274:     Timers.RemoveTimer(TimerID);
275:     pthread_mutex_unlock(&TimerMutex);
276:     //If the deleted timer is the one currently running, cancel the timer
277:     if (TempID == TimerID)
278:     {
279:         itimerval TimeVal;
280:         TimeVal.it_value.tv_sec = 0;

```



```

281:     TimeVal.it_value.tv_usec = 0;
282:
283:     setitimer(ITIMER_REAL, &TimeVal, NULL);
284:
285:     //Start the new timer
286:     pthread_mutex_lock(&TimerMutex);
287:     TempID = Timers.GetSmallestTimerInterval(CurSec, CurUSec, &t1, &t2);
288:     pthread_mutex_unlock(&TimerMutex);
289:     //If no more timers
290:     if (TempID < 0)
291:         return ;
292:     //Otherwise, set this timer
293:     TimeVal.it_value.tv_sec = t1;
294:     TimeVal.it_value.tv_usec = t2;
295:     Timers.SetRunningTimer(TempID);
296:
297:     setitimer(ITIMER_REAL, &TimeVal, NULL);
298: }
299: }
300:
301: #else //VxWorks version - uses POSIX Timers
302: int SDM_CreateTimer(unsigned int Seconds, unsigned int USeconds, void(*Callback)(int), bool
Periodic)
303: {
304:     //Start the signal handling thread
305:     pthread_mutex_lock(&HandlerStartedMutex);
306:     // If the handler needs to be started
307:     if (!HandlerThreadStarted)
308:     {
309:         //Set the global thread mask to ignore SIGALRM
310:         sigset_t SignalMask;
311:         sigemptyset(&SignalMask);
312:         sigaddset(&SignalMask, SIGALRM);
313:         if ((pthread_sigmask(SIG_BLOCK, &SignalMask, NULL)) != 0)
314:         {
315:             perror("SDM_CreateTimer::pthread_sigmask(): ");
316:             pthread_mutex_unlock(&HandlerStartedMutex);
317:             return -1;
318:         }
319:         //Start the HandlerThread
320:         pthread_t HandlerThread;

```

```

321:     if (pthread_create (&HandlerThread, NULL, TimerHandler, NULL) != 0)
322:     {
323:         perror("SDM_CreateTimer::pthread_create(): ");
324:         pthread_mutex_unlock(&HandlerStartedMutex);
325:         return -1;
326:     }
327:     pthread_detach(HandlerThread);
328:     //Now the thread has started, update the HandlerThreadStarted
329:     HandlerThreadStarted = true;
330: }
331: pthread_mutex_unlock(&HandlerStartedMutex);
332:
333: //Get the current time
334: unsigned int CurSec, CurUsec;
335: SDM_GetTime(&CurSec, &CurUsec);
336: //Add the timer to the list
337: pthread_mutex_lock(&TimerMutex);
338:
339: int TimerNum = Timers.AddTimer(Seconds, USeconds, CurSec, CurUsec, (void*)Callback,
Periodic);
340: bool TimerListEmpty = Timers.IsEmpty();
341: //See if there is already a timer running
342:
343: itimerspec TimeVal;
344: timer_gettime(CLOCK_REALTIME, &TimeVal);
345:
346: if (TimerListEmpty || (TimeVal.it_value.tv_sec == 0 && TimeVal.it_value.tv_nsec == 0))
347: {
348:     Timers.SetRunningTimer(TimerNum);
349:     //Start a new timer
350:     TimeVal.it_value.tv_sec = Seconds;
351:     TimeVal.it_value.tv_nsec = USeconds*1000;
352:
353:     timer_settime(CLOCK_REALTIME, 0, &TimeVal, NULL);
354: }
355: //If the current running timer won't expire before this one needs to, reset it to this timer
356: else if (static_cast<unsigned int>(TimeVal.it_value.tv_sec) > Seconds || (static_cast<unsigned
int>(TimeVal.it_value.tv_sec) == Seconds && static_cast<unsigned int>(TimeVal.it_value.tv_nsec) >
USeconds*1000))
357: {
358:     Timers.SetRunningTimer(TimerNum);
359:     TimeVal.it_value.tv_sec = Seconds;

```

```

360:     TimeVal.it_value.tv_nsec = USeconds * 1000;
361:
362:     timer_settime(CLOCK_REALTIME, 0, &TimeVal, NULL);
363: }
364: pthread_mutex_unlock(&TimerMutex);
365: return TimerNum;
366: }
367:
368: /*
369:  * Deletes the timer specified by TimerID. This function has no effect if the timer specified does
370:  * not exist.
371:  *
372:  */
373: void SDM_DeleteTimer(int TimerID)
374: {
375:     //Get the current time
376:     unsigned int CurSec, CurUSec, t1, t2;
377:     SDM_GetTime(&CurSec, &CurUSec);
378:
379:     pthread_mutex_lock(&TimerMutex);
380:     int TempID = Timers.GetRunningTimer();
381:     Timers.RemoveTimer(TimerID);
382:     pthread_mutex_unlock(&TimerMutex);
383:     //If the deleted timer is the one currently running, cancel the timer
384:     if (TempID == TimerID)
385:     {
386:         itimerspec TimeVal;
387:         TimeVal.it_value.tv_sec = 0;
388:         TimeVal.it_value.tv_nsec = 0;
389:
390:         timer_settime(CLOCK_REALTIME, 0, &TimeVal, NULL);
391:
392:         //Start the new timer
393:         pthread_mutex_lock(&TimerMutex);
394:         TempID = Timers.GetSmallestTimerInterval(CurSec, CurUSec, &t1, &t2);
395:         pthread_mutex_unlock(&TimerMutex);
396:         //If no more timers
397:         if (TempID < 0)
398:             return ;
399:         //Otherwise, set this timer
400:         TimeVal.it_value.tv_sec = t1;

```

```

401:     TimeVal.it_value.tv_nsec = t2*1000;
402:     Timers.SetRunningTimer(TempID);
403:
404:     timer_settime(CLOCK_REALTIME, 0, &TimeVal, NULL);
405: }
406: }
407: #endif
408:
409:
410: /*
411:  * Reads the current value for the TimerID specified. The time returned in the output
412:  * parameters is the time remaining until the timer expires.
413:  * Params:
414:  *     TimerID - Identifier number of the timer
415:  *     SecondsOut [OUTPUT] - Pointer to the location where to store the seconds remaining
416:  *     USecondsOut [OUTPUT] - Pointer to the location where to store the micro seconds
417:  *     remaining
418:  * Returns:
419:  *     int - 0 on success, or -1 if some error occurred
420:  */
421: int SDM_ReadTimer(int TimerID, unsigned int *SecondsOut, unsigned int *USecondsOut)
422: {
423:     unsigned int CurSec, CurUSec;
424:     SDM_GetTime(&CurSec, &CurUSec);
425:     //
426:     // Get the time remaining on the timer specified
427:     pthread_mutex_lock(&TimerMutex);
428:     int Result = Timers.GetTimeRemaining(TimerID, CurSec, CurUSec, SecondsOut,
429:     USecondsOut);
430:     pthread_mutex_unlock(&TimerMutex);
431:     return Result;
432: }
433:
434: /*
435:  * Signal handler (for SIGALRM only). Because this thread must lock a mutex, it should be the
436:  * only thread
437:  * to ever handle SIGALRM.
438:  */
439: void* TimerHandler(void *ignored)
440: {

```

```

440:  sigset_t SignalMask;
441:  sigemptyset(&SignalMask);
442:  sigaddset(&SignalMask, SIGALRM);
443:  int SigNum;
444:  while (1)
445:  {
446:      if (sigwait(&SignalMask, &SigNum) != 0)
447:      {
448:          perror ("TimerHandler::sigwait(): ");
449:          return NULL;
450:      }
451:      if (SigNum != SIGALRM) continue;
452:      //Get the current time
453:      unsigned int CurSec, CurUsec;
454:      SDM_GetTime(&CurSec, &CurUsec);
455:      //Find out which timer expired
456:      pthread_mutex_lock(&TimerMutex);
457:      int TimerID = Timers.WhichExpired(CurSec, CurUsec);
458:      if (TimerID < 0)
459:      {
460:          printf("Error (SDMTime::TimerHandler): Could not identify the expired timer. \n");
461:          pthread_mutex_unlock(&TimerMutex);
462:          continue;
463:      }
464:      //Keep processing until all finished timers are done
465:      bool TimerSet = false;
466:      while (TimerID > 0)
467:      {
468:          //Remove the timer if it is finished
469:          if (!Timers.IsPeriodic(TimerID))
470:              Timers.RemoveTimer(TimerID);
471:          //Restart with the minimum timer
472:          unsigned int NewSec, NewUsec;
473:          int NewTimerID = Timers.GetSmallestTimerInterval(CurSec, CurUsec, &NewSec,
474:          &NewUsec);
475: #ifndef __VXWORKS__
476:          itimerval TimerVal;
477:          //See if a timer is currently running
478:          if (!TimerSet)
479:          {

```

```

480:         //If no timer is running, start this one, we can be sure that the first
481:         //entry returned from WhichExpired() is the smallest first
482:         TimerVal.it_value.tv_sec = NewSec;
483:         TimerVal.it_value.tv_usec = NewUSec;
484:         TimerVal.it_interval.tv_sec = 0;
485:         TimerVal.it_interval.tv_usec = 0;
486:         //Set the timer
487:         Timers.SetRunningTimer(NewTimerID);
488:         TimerSet = true;
489:
490:         setitimer(ITIMER_REAL, &TimerVal, NULL);
491:     }
492: #else
493:     itimerspec TimerVal;
494:     //See if a timer is currently running
495:     if (!TimerSet)
496:     {
497:         //If no timer is running, start this one, we can be sure that the first
498:         //entry returned from WhichExpired() is the smallest first
499:         TimerVal.it_value.tv_sec = NewSec;
500:         TimerVal.it_value.tv_nsec = NewUSec*1000;
501:         TimerVal.it_interval.tv_sec = 0;
502:         TimerVal.it_interval.tv_nsec = 0;
503:         //Set the timer
504:         Timers.SetRunningTimer(NewTimerID);
505:         TimerSet = true;
506:
507:         timer_settime(CLOCK_REALTIME, 0, &TimerVal, NULL);
508:     }
509: #endif
510:
511:     //Call the user's callback function
512:     void (*Callback)(int) = (void (*)(int))(Timers.GetCallback(TimerID));
513:     if (Callback == NULL)
514:     {
515:         TimerID = Timers.WhichExpired(CurSec, CurUSec);
516:         continue;
517:     }
518:     (*Callback)(TimerID);
519:     //Get the next timer
520:     TimerID = Timers.WhichExpired(CurSec, CurUSec);

```

```
521:     }
522:     pthread_mutex_unlock(&TimerMutex);
523: }
524: return NULL;
525: }
526:
527: #ifdef __VXWORKS__
528: void usleep(int uSecs)
529: {
530:     int nanoSecs = uSecs * 1000;
531:     struct timespec time;
532:     time.tv_sec = 0;
533:     time.tv_nsec = nanoSecs;
534:     nanosleep(&time, NULL);
535: }
536: #endif
```

File: sdm/common/Time/TimerList.h

```
1: #ifndef _SDM_TIMER_LIST_H_
2: #define _SDM_TIMER_LIST_H_
3: #include "SecTime.h"
4:
5: #define MAX_NUM_TIMERS 5
6: #define MAX_USEC_VALUE 1000000
7: #define TIMER_INACTIVE 0
8: #define TIMER_NOT_FOUND -1
9:
10: struct TimerListItem
11: {
12:     SecTime TimerVal;
13:     SecTime StartTime;
14:     int TimerID;
15:     bool Periodic;
16:     void *Callback;
17:     TimerListItem():TimerVal(),StartTime(),TimerID(0),Periodic(false),Callback(NULL)
18: { }
19: };
20:
21: class SDMLIB_API TimerList
22: {
23: public:
24:     TimerList();
25:     int AddTimer(unsigned int Seconds, unsigned int USeconds, unsigned int CurSeconds, unsigned int CurUSeconds, void *Callback, bool Periodic);
26:     int AddTimer(unsigned int Seconds, unsigned int USeconds, unsigned int CurSeconds, unsigned int CurUSeconds, void *Callback, bool Periodic, unsigned int AssignedID);
27:     int WhichExpired(unsigned int CurSeconds, unsigned int CurUSeconds);
28:     bool RemoveTimer(int TimerID);
29:     void *GetCallback(int TimerID);
30:     bool IsPeriodic(int TimerID);
31:     void TimerFired(int TimerID, unsigned int Seconds, unsigned int USeconds);
32:     int GetSmallestTimerInterval(unsigned int CurSeconds, unsigned int CurUSeconds, unsigned int *TimerSecOut, unsigned int *TimerUsecOut);
33:     int GetTimeRemaining(int TimerID, unsigned int CurSeconds, unsigned int CurUSeconds, unsigned int *TimerSecOut, unsigned int *TimerUsecOut);
34:     bool IsEmpty();
35:     void SetRunningTimer(int TimerID);
36:     int GetRunningTimer() { return CurRunningTimer; }
```



```
37: private:
38: int GetIndexOfID(int ID);
39: int CurID;
40: int CurRunningTimer;
41: TimerListItem Timers[MAX_NUM_TIMERS];
42: };
43:
44: #endif
```

File: sdm/common/Time/Makefile

```
1: #Time makefile
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: CXXFLAGS += -I./
7:
8: .PHONY: all clean distclean
9:
10: all: SDMTimeLinux.o TimerList.o SecTime.o TimeKeepingLinux.o
11:
12: SDMTimeLinux.o: SDMTimeLinux.cpp SDMTimeLinux.h
13: $(CXX) $(CXXFLAGS) $(TIMEFLAGS) -fPIC -c $<
14:
15: TimerList.o: TimerList.cpp TimerList.h
16: $(CXX) $(CXXFLAGS) -fPIC -c $<
17:
18: SecTime.o: SecTime.cpp SecTime.h
19: $(CXX) $(CXXFLAGS) -fPIC -c $<
20:
21: TimeKeepingLinux.o: TimeKeepingLinux.cpp TimeKeeping.h
22: $(CXX) $(CXXFLAGS) $(TIMEFLAGS) -fPIC -c $<
23:
24: clean:
25: rm -f *.o *~
26:
27: distclean: clean
```

File: sdm/common/Time/SDMTimeWin32.h

```
1: #ifndef _SDM_TIME_WIN32_H_
2: #define _SDM_TIME_WIN32_H_
3: #include <windows.h>
4: #ifdef interface
5: # undef interface
6: #endif
7: #include "../sdmLib.h"
8: #include "SecTime.h"
9:
10: extern SDMLIB_API void SDM_TimeInit();
11: extern SDMLIB_API void CALLBACK TimerFired(HWND hwnd, UINT uMsg, UINT idEvent,
DWORD dwTime );
12: extern SDMLIB_API void SDM_GetTime(unsigned int *SecondsOut, unsigned int *USecondsOut);
13: extern SDMLIB_API void SDM_GetTime(SecTime &secTime);
14: extern SDMLIB_API void SDM_Sleep(unsigned int Seconds, unsigned int USeconds);
15: extern SDMLIB_API int SDM_CreateTimer(unsigned int Seconds, unsigned int USeconds,
void(*Callback)(int), bool Periodic);
16: extern SDMLIB_API void SDM_DeleteTimer(int TimerID);
17: extern SDMLIB_API int SDM_ReadTimer(int TimerID, unsigned int *SecondsOut, unsigned int
*USecondsOut);
18:
19: extern SDMLIB_API void *MessageDispatch(void *args);
20:
21:
22: #endif
```

File: sdm/common/Time/TimeKeeping.h

```
1: #ifndef TimeKeeping_h
2: #define TimeKeeping_h
3:
4: #include "../sdmLib.h"
5: // Time keeping values
6: struct SDMLIB_API TimeKeepingRec
7: {
8:   int mode; // 0=unscaled, free running; 1=time scaled, free running; 2 = unscaled, paused, 3 = scaled,
   paused
9:   unsigned long gpsEpochTareSeconds;
10:  double ratio, uSecScale, nSecScale;
11:  long long tareCount, perfCountFreq;
12: };
13: extern struct SDMLIB_API TimeKeepingRec* timeKeeping;
14:
15: extern SDMLIB_API void InitializeTimeKeeping();
16: extern SDMLIB_API struct SDMLIB_API TimeKeepingRec* GetTimeKeeping();
17:
18: //
19: // Defined only for Linux
20: extern bool IsTimeInitialized();
21:
22: #endif
```

File: sdm/common/Time/SDMTimeWin32.cpp

```
1: //-----
2: //-----WIN32 IMPLEMENTATION-----
3: //-----
4: #include <pthread.h>
5: #include <sys/time.h>
6: #include <stdio.h>
7: #include "TimeKeeping.h"
8: #include "SDMTimeWin32.h"
9: #include "TimerList.h"
10: #include "LookupTable.h"
11:
12: static LookupTable IDTable;
13: static pthread_mutex_t IDTableMutex = PTHREAD_MUTEX_INITIALIZER;
14: static TimerList Timers;
15: static pthread_mutex_t TimerMutex = PTHREAD_MUTEX_INITIALIZER;
16: static pthread_t EventReceiver;
17: static pthread_mutex_t ThreadStartedMutex = PTHREAD_MUTEX_INITIALIZER;
18:
19: struct DispatchArgs
20: {
21:     unsigned int TimeoutMs;
22:     int UpdateTableKey;
23: };
24: static bool ThreadStarted = false;
25:
26: /*
27:  * Initializes the timer service. It prevents user program threads from being able to handle
28:  * SIGALRM. This problem is only applicable to the Linux and RTEMS builds.
29:  */
30:
31: void SDM_TimeInit()
32: {
33:     InitializeTimeKeeping();
34: }
35:
36: /*
37:  * Gets the current system time and stores the current seconds in SecondsOut, and the current
38:  * microseconds value in USecondsOut.
39:  * Params:
```

```

40: *      SecondsOut [OUTPUT] - Pointer location to store the current number of seconds
41: *      USecondsOut [OUTPUT] - Pointer location to store the current number of micro seconds
42: */
43: void SDM_GetTime(unsigned int *SecondsOut, unsigned int *USecondsOut)
44: {
45: if (SecondsOut == NULL || USecondsOut == NULL) return;
46: //Get and return the time from Windows
47: LARGE_INTEGER count;
48: QueryPerformanceCounter( &count );
49: long long scaledCount = count.QuadPart - timeKeeping->tareCount;
50: if ( timeKeeping->mode == 1 )
51:     scaledCount = (long long)(scaledCount*timeKeeping->ratio);
52: unsigned long sec = (unsigned long)(scaledCount / timeKeeping->perfCountFreq);
53: long long seconds = sec * timeKeeping->perfCountFreq;
54: *SecondsOut = (unsigned int)(sec + timeKeeping->gpsEpochTareSeconds);
55: *USecondsOut = (unsigned int)((scaledCount-seconds) * timeKeeping->uSecScale);
56: }
57:
58: /*
59: * Gets the current system time and returns it as a SecTime
60: * Params:
61: *      SecTime [INPUT/OUTPUT] - Reference to a SecTime structure
62: */
63: void SDM_GetTime( SecTime &secTime )
64: {
65: //Get and return the time from Windows
66: LARGE_INTEGER count;
67: QueryPerformanceCounter( &count );
68: long long scaledCount = count.QuadPart - timeKeeping->tareCount;
69: if ( timeKeeping->mode == 1 )
70:     scaledCount = (long long)(scaledCount*timeKeeping->ratio);
71: unsigned long sec = (unsigned long)(scaledCount / timeKeeping->perfCountFreq);
72: long long seconds = sec * timeKeeping->perfCountFreq;
73: secTime.SetSeconds( (unsigned int)(sec + timeKeeping->gpsEpochTareSeconds) );
74: secTime.SetUSeconds( (unsigned int)((scaledCount-seconds) * timeKeeping->uSecScale) );
75: }
76:
77: /*
78: * Sleep for the time specified in Seconds and USeconds.
79: * Params:
80: *      Seconds - The number of seconds to sleep

```

```

81: *      USeconds - The number of microseconds to sleep
82: *
83: */
84: void SDM_Sleep(unsigned int Seconds, unsigned int USeconds)
85: {
86:     UINT secondsAsMilliseconds = (UINT)(Seconds * 1000 / timeKeeping->ratio);    //First add the
Seconds
87: //Convert microseconds to milliseconds, rounding up
88:     UINT ms = ((UINT)(USeconds/timeKeeping->ratio+999))/1000;
89:
90:     Sleep(secondsAsMilliseconds+ms);
91: }
92: /*
93: * Create a timer to expire at the specified number of Seconds and USeconds. This function returns
94: * an ID number for the timer to distinguish between multiple timers, and to perform other timer
95: * related operations on the ID. Optionally, the function specified by Callback will be called
96: * when the timer expires.
97: * Params:
98: *     Seconds - The number of seconds until the timer is to expire
99: *     USeconds - The number of microseconds until the timer is to expire
100: *     Callback - The function to call when the timer expires (can be NULL).
101: *     Periodic - Whether the timer should auto-restart upon expiration.
102: * Returns:
103: *     int - An identifier number for the timer.
104: */
105: int SDM_CreateTimer(unsigned int Seconds, unsigned int USeconds, void(*Callback)(int), bool
Periodic)
106: {
107:     int UserID;
108:     pthread_t TimerThread;
109:     unsigned int CurSec, CurUsec;
110:     SDM_GetTime(&CurSec, &CurUsec);
111:     //Convert to milliseconds, if 0 < USeconds < 1000, default it to 1 ms
112:     if (USeconds > 0 && USeconds < 1000)
113:         USeconds = 1;
114:     else
115:         USeconds = static_cast<unsigned int>(USeconds / 1000.0);
116:
117:     UINT TimeInMilliseconds = Seconds * 1000;    //First add the Seconds
118:     TimeInMilliseconds += USeconds;              //Now add the ms
119:     //

```

```

120: // Add an empty value
121: pthread_mutex_lock(&IDTableMutex);
122: UserID = IDTable.AddValue(0);
123: pthread_mutex_unlock(&IDTableMutex);
124: //
125: // Create a timer entry
126: pthread_mutex_lock(&TimerMutex);
127: Timers.AddTimer(Seconds, USeconds, CurSec, CurUSec, (void*)Callback, Periodic, UserID);
128: pthread_mutex_unlock(&TimerMutex);
129: //
130: // Start the timer thread
131: DispatchArgs *TimerThreadArgs = new DispatchArgs();
132: TimerThreadArgs->UpdateTableKey = UserID;
133: TimerThreadArgs->TimeoutMs = TimeInMilliseconds;
134: pthread_create(&TimerThread, NULL, MessageDispatch, TimerThreadArgs);
135: pthread_detach(TimerThread);
136: return UserID;
137: }
138: /*
139:  * Deletes the timer specified by TimerID. This function has no effect if the timer specified does
140:  * not exist.
141:  *
142:  */
143: void SDM_DeleteTimer(int TimerID)
144: {
145: //
146: // First, get the OS timer ID associated with the User timer ID
147: pthread_t TimerThread;
148: bool HaveThreadID = false;
149:
150: pthread_mutex_lock(&IDTableMutex);
151: int OSTimerID = (int)IDTable.GetValueFromKey(TimerID);
152: HaveThreadID = IDTable.GetThreadIDFromKey(TimerID, &TimerThread);
153: pthread_mutex_unlock(&IDTableMutex);
154: if (OSTimerID == 0)
155: {
156:     printf("Could not identify the timer ID. \n");
157:     return;
158: }
159: //
160: // Kill the timer with the OS

```



```

161:  if (!KillTimer(NULL, OSTimerID))
162:      printf("Could not stop the timer. \n");
163:  //
164:  // Remove the timer from the timers list
165:  pthread_mutex_lock(&TimerMutex);
166:  if (!Timers.RemoveTimer((int)TimerID))
167:      printf("Could not identify timer to remove. \n");
168:  pthread_mutex_unlock(&TimerMutex);
169:  //
170:  // Stop the timer thread
171:  if (HaveThreadID)
172:      pthread_cancel(TimerThread);
173:  else
174:      printf("Warning: Could not stop timer thread for TimerID %d. \n",TimerID);
175:  //
176:  // Remove the IDTable entry for the timer
177:  pthread_mutex_lock(&IDTableMutex);
178:  IDTable.RemoveKeyGetValue(TimerID);
179:  pthread_mutex_unlock(&IDTableMutex);
180: }
181: /*
182:  * Reads the current value for the TimerID specified. The time returned in the output
183:  * parameters is the time remaining until the timer expires.
184:  * Params:
185:  *     TimerID - Identifier number of the timer
186:  *     SecondsOut [OUTPUT] - Pointer to the location where to store the seconds remaining
187:  *     USecondsOut [OUTPUT] - Pointer to the location where to store the micro seconds
188:  * Returns:
189:  *     int - 0 on success, or -1 if some error occurred
190:  *
191:  */
192: int SDM_ReadTimer(int TimerID, unsigned int *SecondsOut, unsigned int *USecondsOut)
193: {
194:  //
195:  // Get the current time
196:  unsigned int CurSec, CurUsec;
197:  SDM_GetTime(&CurSec, &CurUsec);
198:  //
199:  // Get current time remaining
200:  pthread_mutex_lock(&TimerMutex);

```

```

201:  int  Result  =  Timers.GetTimeRemaining(TimerID,  CurSec,  CurUSec,  SecondsOut,
USecSecondsOut);
202:  pthread_mutex_unlock(&TimerMutex);
203:  return Result;
204: }
205: /*
206: * The timer event handler function.
207: *
208: */
209: void CALLBACK TimerFired(HWND hwnd, UINT uMsg, UINT idEvent, DWORD dwTime )
210: {
211:  //
212:  // Get the time that the timer fired
213:  unsigned int CurSec, CurUSec;
214:  SDM_GetTime(&CurSec, &CurUSec);
215:  //
216:  // Get the timer table entry
217:  pthread_mutex_lock(&IDTableMutex);
218:  int UserID = IDTable.GetKeyFromValue(idEvent);
219:  pthread_mutex_unlock(&IDTableMutex);
220:  if (UserID < 0)
221:  {
222:      printf("TimerFired:: Could not identify the timer. \n");
223:      return ;
224:  }
225:  //
226:  // Find out which timer was fired
227:  pthread_mutex_lock(&TimerMutex);
228:  void (*Callback)(int) = (void (*)(int))(Timers.GetCallback((int)UserID));
229:  bool IsPeriodic = Timers.IsPeriodic((int)UserID);
230:  //
231:  // Update timer time
232:  Timers.TimerFired(UserID, CurSec, CurUSec);
233:  pthread_mutex_unlock(&TimerMutex);
234:  //
235:  // If the timer is not periodic, remove it
236:  if (!IsPeriodic)
237:      SDM_DeleteTimer(UserID);
238:  //
239:  // Call the user's callback function
240:  if (Callback == NULL)

```

```

241:     return;
242:     (*Callback)(UserID);
243: }
244: /*
245:  * Message dispatch thread for Win32 "signal" or event that a timer has expired.
246:  *
247:  */
248: void *MessageDispatch(void *ThreadArgs)
249: {
250:     DispatchArgs *Arguments = (DispatchArgs*)ThreadArgs;
251:     //
252:     // Create the timer
253:     UINT_PTR TimerID = SetTimer(NULL, NULL, Arguments->TimeoutMs, (TIMERPROC)
TimerFired);
254:     //
255:     // Update the lookup table with the new entry
256:     pthread_t MyThreadID = pthread_self();
257:     pthread_mutex_lock(&IDTableMutex);
258:     IDTable.UpdateValue(Arguments->UpdateTableKey, (unsigned int)TimerID);
259:     IDTable.UpdateThreadIDFromKey(Arguments->UpdateTableKey, MyThreadID);
260:     pthread_mutex_unlock(&IDTableMutex);
261:     //
262:     // Free the arguments
263:     delete Arguments;
264:
265:     MSG msg = { 0, 0, 0, 0 };
266:     while (1)
267:     {
268:         if (GetMessage(&msg, 0, 0, 0) )
269:         {
270:             if (msg.message == WM_TIMER)
271:             {
272:                 DispatchMessage(&msg);
273:             }
274:         }
275:         Sleep(1);
276:     }
277:     return NULL;
278: }

```

File: sdm/common/Time/SecTime.h

```
1: #ifndef _SEC_TIME_H_
2: #define _SEC_TIME_H_
3: #include "../sdmLib.h"
4:
5: #define USEC_MAX_VALUE 1000000
6:
7: //This class is meant to provide abstraction for doing arithmetic operations on time values
8: //that have both seconds and useconds components.
9: class SDMLIB_API SecTime
10: {
11: public:
12: SecTime();
13: SecTime(long Seconds, long USeconds);
14: SecTime(const SecTime& right);
15: SecTime& operator =(const SecTime& right);
16: bool operator==(const SecTime& right) const;
17: bool operator < (const SecTime& right) const;
18: bool operator > (const SecTime& right) const;
19: bool operator <= (const SecTime& right) const;
20: bool operator >= (const SecTime& right) const;
21: const SecTime operator + (const SecTime &right) const;
22: const SecTime operator - (const SecTime &right) const;
23: SecTime& operator -= (const SecTime &right);
24:
25: void SetSeconds(long S) { Sec = S; }
26: void SetUSeconds(long Us) { USec = Us; }
27: long GetSeconds() const { return Sec; }
28: long GetUSeconds() const { return USec; }
29: void Clear() { Sec = 0; USec = 0; }
30: const char* ToString(char *str, int len ) const;
31: private:
32: long Sec; // Number of seconds since Jan 6, 1980 00:00:00
33: long USec;
34: };
35:
36: #endif
```

File: sdm/common/Time/SecTime.cpp

```
1: #include "SecTime.h"
2: #include "unistd.h"
3: #include "stdio.h"
4:
5: SecTime::SecTime(): Sec(0), USec(0)
6: {
7: }
8:
9: SecTime::SecTime(long Seconds, long USeconds): Sec(Seconds), USec(USeconds)
10: {
11: }
12:
13: SecTime::SecTime(const SecTime &right): Sec(right.Sec), USec(right.UsSec)
14: {
15: }
16: /*
17:  * Assignment operator, simply copies the seconds and useconds value to the target object.
18:  */
19: SecTime& SecTime::operator=(const SecTime &right)
20: {
21:     Sec = right.Sec;
22:     USec = right.UsSec;
23:     return *this;
24: }
25: /*
26:  * Equals operator, checks to see if the time values are the same.
27:  */
28: bool SecTime::operator==(const SecTime &right) const
29: {
30:     if (Sec == right.Sec && USec == right.UsSec)
31:         return true;
32:     return false;
33: }
34: /*
35:  * Less than operator, checks to see if the current object is less than the right object.
36:  */
37: bool SecTime::operator < (const SecTime& right) const
38: {
39:     if (Sec < right.Sec)
```

```

40:     return true;
41: if (Sec == right.Sec && USec < right.UsSec)
42:     return true;
43: return false;
44: }
45: /*
46:  * Greater than operator, checks to see if the current object is greater than the right object.
47:  */
48: bool SecTime::operator > (const SecTime& right) const
49: {
50: if (Sec > right.Sec)
51:     return true;
52: if (Sec == right.Sec && USec > right.UsSec)
53:     return true;
54: return false;
55: }
56: /*
57:  * Less than equals operator, checks to see if the current object is less or equal to the right object.
58:  */
59: bool SecTime::operator <= (const SecTime& right) const
60: {
61: if (Sec < right.Sec )
62:     return true;
63: if ( Sec == right.Sec && USec <= right.UsSec)
64:     return true;
65: return false;
66: }
67: /*
68:  * Greater than equals operator, checks to see if the current object is greater than or equal to the right
  object.
69:  */
70: bool SecTime::operator >= (const SecTime& right) const
71: {
72: if (Sec > right.Sec )
73:     return true;
74: if ( Sec == right.Sec && USec >= right.UsSec)
75:     return true;
76: return false;
77: }
78: /*
79:  * Addition operator, performs an addition of the two objects, accounting for useconds overflow.

```

```

80: */
81: const SecTime SecTime::operator + (const SecTime &right) const
82: {
83:     SecTime Result;
84:     Result.SetSeconds(Sec + right.GetSeconds());
85:     if (USec + right.GetUSeconds() >= USEC_MAX_VALUE)
86:     {
87:         Result.SetSeconds(Result.GetSeconds()+1);
88:         Result.SetUSeconds(USec + right.GetUSeconds() - USEC_MAX_VALUE);
89:     }
90:     else
91:         Result.SetUSeconds (USec + right.GetUSeconds());
92:     return Result;
93: }
94: /*
95:  * Subtraction operator, performs a subtraction of the two objects, accounting for useconds carry.
96:  */
97: const SecTime SecTime::operator - (const SecTime &right) const
98: {
99:     SecTime Result;
100:     Result.SetSeconds(Sec - right.GetSeconds());
101:     if (USec- right.GetUSeconds() < 0)
102:     {
103:         Result.SetSeconds(Result.GetSeconds()-1);
104:         Result.SetUSeconds(USEC_MAX_VALUE + USec - right.GetUSeconds());
105:     }
106:     else
107:         Result.SetUSeconds (USec - right.GetUSeconds());
108:     return Result;
109: }
110: /*
111:  * Subtraction operator, performs a subtraction-assign of the two objects, accounting for useconds
    carry.
112:  */
113: SecTime& SecTime::operator -= (const SecTime &right)
114: {
115:     Sec -= right.GetSeconds();
116:     long rhsUSec = right.GetUSeconds();
117:     if (USec - rhsUSec < 0)
118:     {
119:         Sec--;

```

```

120:         USec = USEC_MAX_VALUE + USec - rhsUSec;
121:     }
122:     else
123:         USec = USec - rhsUSec;
124:     return *this;
125: }
126: /*
127:  * Formats as a string. If negative, useconds is corrected
128:  */
129: const char* SecTime::ToString(char *str, int len ) const
130: {
131:     if ( len < 19 )
132:     {
133:         str[0] = 0;
134:         return str;
135:     }
136:
137:     long dSec = Sec;
138:     long dUsec = USec;
139:     if ( dSec < 0 )
140:     {
141:         dSec++;
142:         if ( dSec == 0)
143:         {
144:             *str++ = '-';
145:             len--;
146:         }
147:         dUsec = 1000000 - dUsec;
148:     }
149:     snprintf(str,len,"%ld.%06ld",dSec,dUsec);
150:     return str;
151: }

```


File: sdm/common/Time/SDMTime.h

```
1: #ifndef _SDM_TIME_H_
2: #define _SDM_TIME_H_
3:
4: #ifdef WIN32
5: # include "SDMTimeWin32.h"
6: #elif defined (RTEMS_BUILD)
7: # include "SDMTimeRTEMS.h"
8: #else
9: # include "SDMTimeLinux.h"
10: #endif
11:
12: #endif
```

File: sdm/common/Time/SDMTimeLinux.h

```
1: #ifndef _SDM_TIME_LINUX_H_
2: #define _SDM_TIME_LINUX_H_
3:
4: #include "SecTime.h"
5:
6: void *TimerHandler(void *args);
7: void SDM_TimeInit();
8: void SDM_GetTime(unsigned int *SecondsOut, unsigned int *USecondsOut);
9: void SDM_GetTime(SecTime &secTime);
10: void SDM_Sleep(unsigned int Seconds, unsigned int USeconds);
11: int SDM_CreateTimer(unsigned int Seconds, unsigned int USeconds, void(*Callback)(int), bool
Periodic);
12: void SDM_DeleteTimer(int TimerID);
13: int SDM_ReadTimer(int TimerID, unsigned int *SecondsOut, unsigned int *USecondsOut);
14: #ifdef __VXWORKS__
15: void usleep(int microseconds);
16: #endif
17:
18: #endif
```

File: sdm/common/Time/LookupTable.cpp

```
1: #include "LookupTable.h"
2:
3: LookupTable::LookupTable(): CurKey(1)
4: {
5: }
6:
7: unsigned int LookupTable::GetValueFromKey(int Key)
8: {
9:     for (int i = 0; i < MAX_TABLE_ITEMS; i++)
10:     {
11:         if (TableItems[i].Key == Key)
12:             return TableItems[i].Value;
13:     }
14:     return 0;
15: }
16:
17: int LookupTable::GetKeyFromValue(unsigned int Value)
18: {
19:     for (int i = 0; i < MAX_TABLE_ITEMS; i++)
20:     {
21:         if (TableItems[i].Value == Value)
22:             return TableItems[i].Key;
23:     }
24:     return -1;
25: }
26:
27: int LookupTable::AddValue(unsigned int Value)
28: {
29:     for (int i = 0; i < MAX_TABLE_ITEMS; i++)
30:         if (TableItems[i].Key == KEY_EMPTY)
31:         {
32:             TableItems[i].Key = CurKey++;
33:             TableItems[i].Value = Value;
34:             return TableItems[i].Key;
35:         }
36:     return -1;
37: }
38:
39: unsigned int LookupTable::RemoveKeyGetValue(int Key)
```

```

40: {
41: unsigned int Result;
42: for (int i = 0; i < MAX_TABLE_ITEMS; i++)
43:     if (TableItems[i].Key == Key)
44:     {
45:         TableItems[i].Key = KEY_EMPTY;
46:         Result = TableItems[i].Value;
47:         TableItems[i].Value = 0;
48:         return Result;
49:     }
50: return 0;
51: }
52:
53: int LookupTable::RemoveValueGetKey(unsigned int Value)
54: {
55: int Result;
56: for (int i = 0; i < MAX_TABLE_ITEMS; i++)
57:     if (TableItems[i].Value == Value)
58:     {
59:         Result = TableItems[i].Key;
60:         TableItems[i].Key = KEY_EMPTY;
61:         TableItems[i].Value = 0;
62:         return Result;
63:     }
64: return 0;
65: }
66:
67:
68: void LookupTable::UpdateValue(int Key, unsigned int NewValue)
69: {
70: for (int i = 0; i < MAX_TABLE_ITEMS; i++)
71:     if (TableItems[i].Key == Key)
72:     {
73:         TableItems[i].Value = NewValue;
74:         return ;
75:     }
76: }
77:
78: bool LookupTable::UpdateThreadIDFromKey(int Key, pthread_t ThreadID)
79: {
80: for (int i = 0; i < MAX_TABLE_ITEMS; i++)

```

```

81: {
82:     if (TableItems[i].Key == Key)
83:     {
84:         TableItems[i].ThreadID = ThreadID;
85:         return true;
86:     }
87: }
88: return false;
89: }
90:
91: bool LookupTable::UpdateThreadIDFromValue(unsigned int Value, pthread_t ThreadID)
92: {
93:     for (int i = 0; i < MAX_TABLE_ITEMS; i++)
94:     {
95:         if (TableItems[i].Value == Value)
96:         {
97:             TableItems[i].ThreadID = ThreadID;
98:             return true;
99:         }
100:     }
101:     return false;
102: }
103:
104: bool LookupTable::GetThreadIDFromKey(int Key, pthread_t *ThreadIDOut)
105: {
106:     for (int i = 0; i < MAX_TABLE_ITEMS; i++)
107:     {
108:         if (TableItems[i].Key == Key)
109:         {
110:             *ThreadIDOut = TableItems[i].ThreadID;
111:             return true;
112:         }
113:     }
114:     return false;
115: }

```

File: sdm/common/MessageManipulator/msgdef.y

```
1: %{
2: /*xTEDS 1.0 msg_def parser*/
3: #include <stdio.h>
4: #include <stdlib.h>
5: #include "../message.h"
6:
7: int yylex(void);
8:
9: void yyerror(const char *str);
10:
11: int yydebug=0;
12:
13: %}
14:
15: /*punctuation*/
16: %token LT_SY CMD_MSG_SY DATA_MSG_SY NAME_SY EQUAL_SY SLASH_SY GT_SY
COLON_SY VAR_SY
17: %token LENGTH_SY COUNTMAX_SY RATEMAX_SY RQST_SY REPLY_SY ID_SY
FAULT_MSG_SY
18: %token CMD_SY NOTI_SY ID_SY DATATYPE FLOAT INT STRING NOT_SPECIFIED_SY
19:
20: %union
21: {
22: int integer;
23: float real;
24: char* str;
25: struct variable* var;
26: struct node_data* node;
27: }
28:
29: %type <str> STRING DATATYPE
30: %type <integer> INT ATTRIBUTE
31: %type <real> FLOAT
32: %type <node> MSG_DEF FAULT_MSG
33: %type <var> VARIABLES VARIABLE
34:
35: %start INFO
36:
37:
```

```

38: %%
39: INFO      : LT_SY RQST_SY GT_SY LT_SY MSG_DEF SLASH_SY GT_SY LT_SY
MSG_DEF SLASH_SY GT_SY LT_SY SLASH_SY RQST_SY GT_SY
40:    {
41:        setMessage($5,$9,NULL);
42:    }
43:    | LT_SY RQST_SY GT_SY LT_SY MSG_DEF SLASH_SY GT_SY LT_SY MSG_DEF
SLASH_SY GT_SY LT_SY FAULT_MSG SLASH_SY GT_SY LT_SY SLASH_SY RQST_SY
GT_SY
44:    {
45:        setMessage($5,$9,$13);
46:    }
47:    | LT_SY CMD_SY GT_SY LT_SY MSG_DEF SLASH_SY GT_SY LT_SY SLASH_SY
CMD_SY GT_SY
48:    {
49:        setMessage($5,NULL,NULL);
50:    }
51:    | LT_SY CMD_SY GT_SY LT_SY MSG_DEF SLASH_SY GT_SY LT_SY FAULT_MSG
SLASH_SY GT_SY LT_SY SLASH_SY CMD_SY GT_SY
52:    {
53:        setMessage($5,NULL,$9);
54:    }
55:    | LT_SY NOTI_SY GT_SY LT_SY MSG_DEF SLASH_SY GT_SY LT_SY SLASH_SY
NOTI_SY GT_SY
56:    {
57:        setMessage(NULL,$5,NULL);
58:    }
59:    | LT_SY NOTI_SY GT_SY LT_SY MSG_DEF SLASH_SY GT_SY LT_SY FAULT_MSG
SLASH_SY GT_SY LT_SY SLASH_SY NOTI_SY GT_SY
60:    {
61:        setMessage(NULL,$5,$9);
62:    }
63:    ;
64:
65: FAULT_MSG : FAULT_MSG_SY NAME_SY EQUAL_SY STRING ID_SY EQUAL_SY
STRING VARIABLES ATTRIBUTE
66:    {
67:        $$ = newNode($4,FAULTMSG,$9,atoi($7),$8);
68:        if ($7 != NULL) free($7);
69:    }
70:    ;
71:

```

```

72: MSG_DEF      :CMD_MSG_SY NAME_SY EQUAL_SY STRING ID_SY EQUAL_SY STRING
VARIABLES ATTRIBUTE
73:      {
74:          $$ = newNode($4,COMMANDMSG,$9,atoi($7),$8);
75:          if ($7 != NULL) free($7);
76:      }
77: |DATA_MSG_SY NAME_SY EQUAL_SY STRING ID_SY EQUAL_SY STRING
VARIABLES ATTRIBUTE
78:      {
79:          $$ = newNode($4,DATAMSG,$9,atoi($7),$8);
80:          if ($7 != NULL) free($7);
81:      }
82:      ;
83:
84: VARIABLES      : VARIABLES VARIABLE
85:      {
86:          $$ = addVar(&$1,$2);
87:      }
88:      /*empty*/
89:      {
90:          $$ = NULL;
91:      }
92:      ;
93:
94: VARIABLE      : VAR_SY COLON_SY INT COLON_SY INT COLON_SY FLOAT COLON_SY
NOT_SPECIFIED_SY COLON_SY DATATYPE EQUAL_SY STRING
95:      { /*Invalid type is not specified*/
96:          $$ = newVar($13,$3,$5,$11,$7);
97:      }
98: | VAR_SY COLON_SY INT COLON_SY INT COLON_SY FLOAT COLON_SY INT
COLON_SY DATATYPE EQUAL_SY STRING
99:      { /*Invalid type is an int*/
100:          $$ = newVarInvalidIsInt($13,$3,$5,$11,$7,$9);
101:      }
102: | VAR_SY COLON_SY INT COLON_SY INT COLON_SY FLOAT COLON_SY FLOAT
COLON_SY DATATYPE EQUAL_SY STRING
103:      { /*Invalid type is a float*/
104:          $$ = newVarInvalidIsFloat($13,$3,$5,$11,$7,$9);
105:      }
106:
107:      ;
108:

```



```
109:
110: ATTRIBUTE : LENGTH_SY EQUAL_SY STRING
111:     {
112:         $$ = atoi($3);
113:         if ($3 != NULL) free($3);
114:     }
115: ;
116: %%
```

File: sdm/common/MessageManipulator/message.h

```
1: #ifndef __MESSAGE_H_
2: #define __MESSAGE_H_
3:
4: enum value_type
5: {
6:  _EMPTY,_UINT08,_INT08,_UINT16,_INT16,_UINT32,_INT32,_FLOAT32,_FLOAT64
7: };
8:
9: union _value
10: {
11:  unsigned char uint08;
12:  char int08;
13:  unsigned short uint16;
14:  short int16;
15:  unsigned long uint32;
16:  signed long int32;
17:  float float32;
18:  double float64;
19: };
20:
21: typedef struct value_s
22: {
23:  enum value_type type;
24:  union _value value;
25: }value_t;
26:
27: typedef struct variable
28: {
29:  char* name;
30:  int start;
31:  int length;
32:  enum value_type type;
33:  float scale_factor;
34:  value_t invalid_data;
35:  struct variable* left;
36:  struct variable* right;
37: }var;
38:
39: enum msg_type
```

```

40: {
41: COMMANDMSG, DATAMSG, FAULTMSG
42: };
43:
44: typedef struct node_data
45: {
46: char* name;
47: enum msg_type type;
48: int length,countmax,id;
49: float ratemax;
50: var* list;
51: }nodeData;
52:
53: enum wrapper_type
54: {
55: EMPTY, COMMAND, NOTIFICATION, REQUEST
56: };
57:
58: typedef struct msg_data
59: {
60: enum wrapper_type type;
61: nodeData* command;
62: nodeData* data;
63: nodeData* fault;
64: }msg;
65:
66: void setLength(nodeData*,int);
67: var* addVar(var** root,var* new_var);
68: var* newVar(char* p_Name,int p_Start,int p_Length,char* p_Type,float p_ScaleFactor);
69: var* newVarInvalidIsInt(char* p_Name, int p_Start, int p_Length, char* p_Type, float p_ScaleFactor,
int p_InvalidValue);
70: var* newVarInvalidIsFloat(char* p_Name, int p_Start, int p_Length, char* p_Type, float
p_ScaleFactor, float p_InvalidValue);
71: nodeData* newNode(char*,enum msg_type,int,int,var*);
72: void setName(nodeData*,char*);
73: void setMessage(nodeData*, nodeData*, nodeData*);
74: int deleteVarTree( var** p);
75: void deleteMessage(msg*);
76:
77: void printMsg();
78: var* find(char* name,var* root);

```

```
79: msg* parse(char* msg_def,msg*);  
80:  
81: #endif
```

File: sdm/common/MessageManipulator/message.c

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <string.h>
4: #include "message.h"
5:
6:
7: msg* Message;
8:
9: int clear( var** p);
10: void PrintVar( var* varlist);
11:
12: void setLength( nodeData* node, int l)
13: {
14:     node->length = l;
15: }
16:
17: var* find(char* name, var* root)
18: {
19:     if (root==NULL || name==NULL) return NULL;
20:     if (strcmp(name,root->name)==0)
21:         return root;
22:     if (strcmp(name,root->name)<0)
23:         return find(name,root->left);
24:     return find(name,root->right);
25: }
26:
27: var* addVar(var** root, var* new_var)
28: {
29:     if (new_var==NULL) return *root;
30:     if(*root==NULL)
31:     {
32:         *root=new_var;
33:         return *root;
34:     }
35:     if(strcmp(new_var->name,(*root)->name)<0)
36:     {
37:         addVar(&((*root)->left),new_var);
38:         return *root;
39:     }
```

```

40: else
41: {
42:     addVar(&((*root)->right),new_var);
43:     return *root;
44: }
45: }
46: /*
47: Overload to handle a variable's invalid value of type int.
48: */
49: var* newVarInvalidIsInt(char* p_Name, int p_Start, int p_Length, char* p_Type, float p_ScaleFactor,
int p_InvalidValue)
50: {
51: var* NewVar = newVar(p_Name, p_Start, p_Length, p_Type, p_ScaleFactor);
52: if (NewVar == NULL)
53:     return NULL;
54:
55: NewVar->invalid_data.type = NewVar->type;
56: switch(NewVar->type)
57: {
58:     case _UINT08:
59:         NewVar->invalid_data.value.uint08 = (unsigned char)p_InvalidValue;
60:         break;
61:     case _INT08:
62:         NewVar->invalid_data.value.int08 = (char)p_InvalidValue;
63:         break;
64:     case _UINT16:
65:         NewVar->invalid_data.value.uint16 = (unsigned short)p_InvalidValue;
66:         break;
67:     case _INT16:
68:         NewVar->invalid_data.value.int16 = (short)p_InvalidValue;
69:         break;
70:     case _UINT32:
71:         NewVar->invalid_data.value.uint32 = (unsigned int)p_InvalidValue;
72:         break;
73:     case _INT32:
74:         NewVar->invalid_data.value.int32 = p_InvalidValue;
75:         break;
76:     case _FLOAT32:
77:         NewVar->invalid_data.value.float32 = (float)p_InvalidValue;
78:         break;
79:     case _FLOAT64:

```

```

80:         NewVar->invalid_data.value.float64 = (double)p_InvalidValue;
81:         break;
82:     case _EMPTY:
83:         break;
84: }
85: return NewVar;
86: }
87: /*
88: Overload to handle a variable's invalid value of type float.
89: */
90: var* newVarInvalidIsFloat(char* p_Name, int p_Start, int p_Length, char* p_Type, float
p_ScaleFactor, float p_InvalidValue)
91: {
92: var* NewVar = newVar(p_Name, p_Start, p_Length, p_Type, p_ScaleFactor);
93: if (NewVar == NULL)
94:     return NULL;
95:
96: NewVar->invalid_data.type = NewVar->type;
97: switch(NewVar->type)
98: {
99:     case _FLOAT32:
100:         NewVar->invalid_data.value.float32 = p_InvalidValue;
101:         break;
102:     case _FLOAT64:
103:         NewVar->invalid_data.value.float64 = (double)p_InvalidValue;
104:         break;
105:     default:
106:         printf("MessageManipulator::Warning - Variable %s received floating point invalid value
of %f but is of type %s. \n", p_Name, p_InvalidValue, p_Type);
107:         break;
108: }
109: return NewVar;
110: }
111: /*
112: Allocates a new variable and sets its name, starting location, length, type, and scale factor.
113: */
114: var* newVar(char* p_Name,int p_Start,int p_Length,char* p_Type,float p_ScaleFactor)
115: {
116:     var* temp;
117:     temp = ( var*)malloc(sizeof( var));
118:     memset(temp, 0, sizeof(var));

```

```

119: temp->name = p_Name;
120: temp->start = p_Start;
121: temp->length = p_Length;
122:
123: /* Set the type */
124: if (strcmp(p_Type,"UINT08")==0)
125:     temp->type = _UINT08;
126: else if (strcmp(p_Type,"INT08")==0)
127:     temp->type = _INT08;
128: else if (strcmp(p_Type,"UINT16")==0)
129:     temp->type = _UINT16;
130: else if (strcmp(p_Type,"INT16")==0)
131:     temp->type = _INT16;
132: else if (strcmp(p_Type,"UINT32")==0)
133:     temp->type = _UINT32;
134: else if (strcmp(p_Type,"INT32")==0)
135:     temp->type = _INT32;
136: else if (strcmp(p_Type,"FLOAT32")==0)
137:     temp->type = _FLOAT32;
138: else if (strcmp(p_Type,"FLOAT64")==0)
139:     temp->type = _FLOAT64;
140:
141: free(p_Type);
142: temp->invalid_data.type = _EMPTY;
143: temp->scale_factor = p_ScaleFactor;
144: temp->left = NULL;
145: temp->right = NULL;
146: return temp;
147: }
148:
149: nodeData* newNode(char* name,enum msg_type type,int length,int id, var* list)
150: {
151:     nodeData* temp;
152:     temp = (nodeData*)malloc(sizeof(nodeData));
153:     memset(temp, 0, sizeof(nodeData));
154:     setName(temp, name);
155:     temp->type = type;
156:     setLength(temp,length);
157:     temp->id = id;
158:     temp->list = list;
159:     return temp;

```



```

160: }
161:
162: void setName( nodeData* node, char* n)
163: {
164:     node->name = n;
165: }
166:
167: void setMessage( nodeData* command, nodeData* data, nodeData* fault)
168: {
169:     deleteMessage(Message);
170:     /* Assign the new message nodes */
171:     Message->command = command;
172:     Message->data = data;
173:     Message->fault = fault;
174:     /* Assign the message type */
175:     if(Message->data == NULL)
176:         Message->type = COMMAND;
177:     else if(Message->command == NULL)
178:         Message->type = NOTIFICATION;
179:     else
180:         Message->type = REQUEST;
181: }
182:
183: void printMsg()
184: {
185:     switch(Message->type)
186:     {
187:         case COMMAND:
188:             printf("Message type: COMMAND \n");
189:             break;
190:         case NOTIFICATION:
191:             printf("Message type: NOTIFICATION \n");
192:             break;
193:         case REQUEST:
194:             printf("Message type: COMMAND \n");
195:             break;
196:         case EMPTY:
197:             printf("Message type: Message Def has not been parsed yet! \n");
198:             return;
199:     }
200:     if(Message->command != NULL)

```

```

201:     printf("CommandMsg name: %s id: %x length %d \n",Message->command->name,Message-
>command->id,Message->command->length);
202:     if(Message->data != NULL)
203:         printf("DataMsg name: %s id: %x length %d \n",Message->data->name,Message->data-
>id,Message->data->length);
204:     if(Message->command != NULL)
205:         printf("FaultMsg name: %s id: %x length %d \n",Message->fault->name,Message->fault-
>id,Message->fault->length);
206: }
207:
208: void PrintVar( var* varlist)
209: {
210:     if(varlist == NULL)
211:         return;
212:     printf("Variable name: %s type: %d start: %d length: %d \n",varlist->name,varlist->type,varlist-
>start,varlist->length);
213:     PrintVar(varlist->left);
214:     PrintVar(varlist->right);
215: }
216:
217: msg* parse(char* msg_def, msg* msg_ptr)
218: {
219:     extern void *MessageManipulator_scan_string(const char* str);
220:     extern int MessageManipulatorparse();
221:     extern void MessageManipulator_delete_buffer(void*);
222:     void* buffer = NULL;
223:     int parseResult;
224:
225:     Message = msg_ptr;
226:     /* Parse the new message defs */
227:     buffer = MessageManipulator_scan_string(msg_def);
228:     parseResult = MessageManipulatorparse();
229:     MessageManipulator_delete_buffer(buffer);
230:     if (parseResult==0)
231:     {
232:         return Message;
233:     }
234:     return NULL;
235: }
236:
237: void deleteMessage(msg* Msg)
238: {

```

```

239:   if (Msg == NULL) return;
240:   /* Delete the command message list */
241:   if(Msg->command != NULL)
242:   {
243:       if (Msg->command->name != NULL)
244:           free (Msg->command->name);
245:       deleteVarTree(&(Msg->command->list));
246:       free (Msg->command);
247:       Msg->command = NULL;
248:   }
249:   /* Delete the data message list */
250:   if(Msg->data != NULL)
251:   {
252:       if (Msg->data->name != NULL)
253:           free (Msg->data->name);
254:       deleteVarTree(&(Msg->data->list));
255:       free (Msg->data);
256:       Msg->data = NULL;
257:   }
258:   /*Delete the fault message list*/
259:   if(Msg->fault != NULL)
260:   {
261:       if (Msg->fault->name != NULL)
262:           free (Msg->fault->name);
263:       deleteVarTree(&(Msg->fault->list));
264:       free (Msg->fault);
265:       Msg->fault = NULL;
266:   }
267: }
268:
269: int deleteVarTree( var** p)
270: {
271:     if((*p)==NULL) return 0;
272:
273:     /*Free the heap allocated members*/
274:     if ((*p)->name != NULL)
275:         free ((*p)->name);
276:
277:     /*Recursively delete the other tree variable members*/
278:     if((*p)->left!=*p)
279:         deleteVarTree(&((*p)->left));

```

```
280:  if((*p)->right!=*p)
281:      deleteVarTree(&((*p)->right));
282:  free(*p);
283:  *p=NULL;
284:  return 1;
285: }
```

File: sdm/common/MessageManipulator/msgdef.tab.c

```
1: /* A Bison parser, made by GNU Bison 1.875d. */
2:
3: /* Skeleton parser for Yacc-like parsing with Bison,
4:  Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.
5:
6:  This program is free software; you can redistribute it and/or modify
7:  it under the terms of the GNU General Public License as published by
8:  the Free Software Foundation; either version 2, or (at your option)
9:  any later version.
10:
11:  This program is distributed in the hope that it will be useful,
12:  but WITHOUT ANY WARRANTY; without even the implied warranty of
13:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14:  GNU General Public License for more details.
15:
16:  You should have received a copy of the GNU General Public License
17:  along with this program; if not, write to the Free Software
18:  Foundation, Inc., 59 Temple Place - Suite 330,
19:  Boston, MA 02111-1307, USA. */
20:
21: /* As a special exception, when this file is copied by Bison into a
22:  Bison output file, you may use that output file without restriction.
23:  This special exception was added by the Free Software Foundation
24:  in version 1.24 of Bison. */
25:
26: /* Written by Richard Stallman by simplifying the original so called
27:  ``semantic" parser. */
28:
29: /* All symbols defined below should begin with yy or YY, to avoid
30:  infringing on user name space.  This should be done even for local
31:  variables, as they might otherwise be expanded by user macros.
32:  There are some unavoidable exceptions within include files to
33:  define necessary library symbols; they are noted "INFRINGES ON
34:  USER NAME SPACE" below. */
35:
36: /* Identify Bison output. */
37: #define YYBISON 1
38:
39: /* Skeleton name. */
```

```

40: #define YYSKELETON_NAME "yacc.c"
41:
42: /* Pure parsers. */
43: #define YYPURE 0
44:
45: /* Using locations. */
46: #define YYLSP_NEEDED 0
47:
48: /* If NAME_PREFIX is specified substitute the variables and functions
49:    names. */
50: #define yyparse MessageManipulatorparse
51: #define yylex MessageManipulatorlex
52: #define yyerror MessageManipulatorerror
53: #define yylval MessageManipulatorlval
54: #define yychar MessageManipulatorchar
55: #define yydebug MessageManipulatordebug
56: #define yynerrs MessageManipulatornerrs
57:
58:
59: /* Tokens. */
60: #ifndef YYTOKENTYPE
61: # define YYTOKENTYPE
62: /* Put the tokens into the symbol table, so that GDB and other debuggers
63:    know about them. */
64: enum yytokentype {
65:    LT_SY = 258,
66:    CMD_MSG_SY = 259,
67:    DATA_MSG_SY = 260,
68:    NAME_SY = 261,
69:    EQUAL_SY = 262,
70:    SLASH_SY = 263,
71:    GT_SY = 264,
72:    COLON_SY = 265,
73:    VAR_SY = 266,
74:    LENGTH_SY = 267,
75:    COUNTMAX_SY = 268,
76:    RATEMAX_SY = 269,
77:    RQST_SY = 270,
78:    REPLY_SY = 271,
79:    ID_SY = 272,
80:    FAULT_MSG_SY = 273,

```

```

81:  CMD_SY = 274,
82:  NOTI_SY = 275,
83:  DATATYPE = 276,
84:  FLOAT = 277,
85:  INT = 278,
86:  STRING = 279,
87:  NOT_SPECIFIED_SY = 280
88:  };
89: #endif
90: #define LT_SY 258
91: #define CMD_MSG_SY 259
92: #define DATA_MSG_SY 260
93: #define NAME_SY 261
94: #define EQUAL_SY 262
95: #define SLASH_SY 263
96: #define GT_SY 264
97: #define COLON_SY 265
98: #define VAR_SY 266
99: #define LENGTH_SY 267
100: #define COUNTMAX_SY 268
101: #define RATEMAX_SY 269
102: #define RQST_SY 270
103: #define REPLY_SY 271
104: #define ID_SY 272
105: #define FAULT_MSG_SY 273
106: #define CMD_SY 274
107: #define NOTI_SY 275
108: #define DATATYPE 276
109: #define FLOAT 277
110: #define INT 278
111: #define STRING 279
112: #define NOT_SPECIFIED_SY 280
113:
114:
115:
116:
117: /* Copy the first part of user declarations. */
118: #line 1 "msgdef.y"
119:
120: /*xTEDS 1.0 msg_def parser*/
121: #include <stdio.h>

```

```

122: #include <stdlib.h>
123: #include "../message.h"
124:
125: int yylex(void);
126:
127: void yyerror(const char *str);
128:
129: int yydebug=0;
130:
131:
132:
133: /* Enabling traces. */
134: #ifndef YYDEBUG
135: # define YYDEBUG 1
136: #endif
137:
138: /* Enabling verbose error messages. */
139: #ifdef YYERROR_VERBOSE
140: # undef YYERROR_VERBOSE
141: # define YYERROR_VERBOSE 1
142: #else
143: # define YYERROR_VERBOSE 0
144: #endif
145:
146: #if ! defined (YYSTYPE) && ! defined (YYSTYPE_IS_DECLARED)
147: #line 21 "msgdef.y"
148: typedef union YYSTYPE {
149:     int integer;
150:     float real;
151:     char* str;
152:     struct variable* var;
153:     struct node_data* node;
154: } YYSTYPE;
155: /* Line 191 of yacc.c. */
156: #line 157 "msgdef.tab.c"
157: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
158: # define YYSTYPE_IS_DECLARED 1
159: # define YYSTYPE_IS_TRIVIAL 1
160: #endif
161:
162:

```



```

163:
164: /* Copy the second part of user declarations. */
165:
166:
167: /* Line 214 of yacc.c. */
168: #line 169 "msgdef.tab.c"
169:
170: #if ! defined (yyoverflow) || YYERROR_VERBOSE
171:
172: # ifndef YYFREE
173: #  define YYFREE free
174: # endif
175: # ifndef YYMALLOC
176: #  define YYMALLOC malloc
177: # endif
178:
179: /* The parser invokes alloca or malloc; define the necessary symbols. */
180:
181: # ifdef YYSTACK_USE_ALLOCA
182: #  if YYSTACK_USE_ALLOCA
183: #   define YYSTACK_ALLOC alloca
184: #  endif
185: # else
186: #  if defined (alloca) || defined (_ALLOCA_H)
187: #   define YYSTACK_ALLOC alloca
188: #  else
189: #   ifdef __GNUC__
190: #    define YYSTACK_ALLOC __builtin_alloca
191: #   endif
192: #  endif
193: # endif
194:
195: # ifdef YYSTACK_ALLOC
196: /* Pacify GCC's 'empty if-body' warning. */
197: #  define YYSTACK_FREE(Ptr) do { /* empty */; } while (0)
198: # else
199: #  if defined (__STDC__) || defined (__cplusplus)
200: #   include <stdlib.h> /* INFRINGES ON USER NAME SPACE */
201: #   define YYSIZE_T size_t
202: #  endif
203: #  define YYSTACK_ALLOC YYMALLOC

```

```

204: # define YYSTACK_FREE YYFREE
205: # endif
206: #endif /* ! defined (yyoverflow) || YYERROR_VERBOSE */
207:
208:
209: #if (! defined (yyoverflow) \
210:      && (! defined (__cplusplus) \
211:          || (defined (YYSTYPE_IS_TRIVIAL) && YYSTYPE_IS_TRIVIAL)))
212:
213: /* A type that is properly aligned for any stack member. */
214: union yyallocl
215: {
216:   short int yyss;
217:   YYSTYPE yyvs;
218: };
219:
220: /* The size of the maximum gap between one aligned stack and the next. */
221: # define YYSTACK_GAP_MAXIMUM (sizeof (union yyallocl) - 1)
222:
223: /* The size of an array large to enough to hold all stacks, each with
224:    N elements. */
225: # define YYSTACK_BYTES(N) \
226:   ((N) * (sizeof (short int) + sizeof (YYSTYPE)) \
227:    + YYSTACK_GAP_MAXIMUM)
228:
229: /* Copy COUNT objects from FROM to TO. The source and destination do
230:    not overlap. */
231: # ifndef YYCOPY
232: #   if defined (__GNUC__) && 1 < __GNUC__
233: #     define YYCOPY(To, From, Count) \
234:       __builtin_memcpy (To, From, (Count) * sizeof (*(From)))
235: #   else
236: #     define YYCOPY(To, From, Count) \
237:       do \
238:       { \
239:         register YYSIZE_T yyi; \
240:         for (yyi = 0; yyi < (Count); yyi++) \
241:           (To)[yyi] = (From)[yyi]; \
242:       } \
243:       while (0)
244: #   endif

```

```

245: # endif
246:
247: /* Relocate STACK from its old location to the new one. The
248:  local variables YYSIZE and YYSTACKSIZE give the old and new number of
249:  elements in the stack, and YYPTR gives the new location of the
250:  stack. Advance YYPTR to a properly aligned location for the next
251:  stack. */
252: # define YYSTACK_RELOCATE(Stack)          \
253:  do                                       \
254:  {                                       \
255:    YYSIZE_T yynewbytes;                  \
256:    YYCOPY (&yyptr->Stack, Stack, yysize); \
257:    Stack = &yyptr->Stack;                 \
258:    yynewbytes = yystacksize * sizeof (*Stack) + YYSTACK_GAP_MAXIMUM; \
259:    yyptr += yynewbytes / sizeof (*yyptr); \
260:  }                                       \
261:  while (0)
262:
263: #endif
264:
265: #if defined (__STDC__) || defined (__cplusplus)
266:  typedef signed char yysigned_char;
267: #else
268:  typedef short int yysigned_char;
269: #endif
270:
271: /* YYFINAL -- State number of the termination state. */
272: #define YYFINAL 6
273: /* YYLAST -- Last index in YYTABLE. */
274: #define YYLAST 123
275:
276: /* YYNTOKENS -- Number of terminals. */
277: #define YYNTOKENS 26
278: /* YYNNTS -- Number of nonterminals. */
279: #define YYNNTS 7
280: /* YYNRULES -- Number of rules. */
281: #define YYNRULES 16
282: /* YYNSTATES -- Number of states. */
283: #define YYNSTATES 113
284:
285: /* YYTRANSLATE(YYLEX) -- Bison symbol number corresponding to YYLEX. */

```

```

286: #define YYUNDEFTOK 2
287: #define YYMAXUTOK 280
288:
289: #define YYTRANSLATE(YYX) \
290: ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)
291:
292: /* YYTRANSLATE[YYLEX] -- Bison symbol number corresponding to YYLEX. */
293: static const unsigned char yytranslate[] =
294: {
295:     0,  2,  2,  2,  2,  2,  2,  2,  2,  2,
296:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
297:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
298:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
299:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
300:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
301:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
302:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
303:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
304:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
305:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
306:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
307:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
308:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
309:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
310:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
311:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
312:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
313:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
314:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
315:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
316:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
317:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
318:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
319:     2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
320:     2,  2,  2,  2,  2,  2,  1,  2,  3,  4,
321:     5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
322:    15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
323:    25
324: };
325:
326: #if YYDEBUG

```

```

327: /* YYPRHS[YYN] -- Index of the first RHS symbol of rule number YYN in
328:   YYRHS. */
329: static const unsigned char yyprhs[] =
330: {
331:     0,  0,  3, 19, 39, 51, 67, 79, 95, 105,
332:    115, 125, 128, 129, 143, 157, 171
333: };
334:
335: /* YYRHS -- A '-1'-separated list of the rules' RHS. */
336: static const yysigned_char yyrhs[] =
337: {
338:     27,  0, -1,  3, 15,  9,  3, 29,  8,  9,
339:     3, 29,  8,  9,  3,  8, 15,  9, -1,  3,
340:    15,  9,  3, 29,  8,  9,  3, 29,  8,  9,
341:     3, 28,  8,  9,  3,  8, 15,  9, -1,  3,
342:    19,  9,  3, 29,  8,  9,  3,  8, 19,  9,
343:    -1,  3, 19,  9,  3, 29,  8,  9,  3, 28,
344:     8,  9,  3,  8, 19,  9, -1,  3, 20,  9,
345:     3, 29,  8,  9,  3,  8, 20,  9, -1,  3,
346:    20,  9,  3, 29,  8,  9,  3, 28,  8,  9,
347:     3,  8, 20,  9, -1, 18,  6,  7, 24, 17,
348:     7, 24, 30, 32, -1,  4,  6,  7, 24, 17,
349:     7, 24, 30, 32, -1,  5,  6,  7, 24, 17,
350:     7, 24, 30, 32, -1, 30, 31, -1, -1, 11,
351:    10, 23, 10, 23, 10, 22, 10, 25, 10, 21,
352:     7, 24, -1, 11, 10, 23, 10, 23, 10, 22,
353:    10, 23, 10, 21,  7, 24, -1, 11, 10, 23,
354:    10, 23, 10, 22, 10, 22, 10, 21,  7, 24,
355:    -1, 12,  7, 24, -1
356: };
357:
358: /* YYRLINE[YYN] -- source line where rule number YYN was defined. */
359: static const unsigned char yyrline[] =
360: {
361:     0, 39, 39, 43, 47, 51, 55, 59, 65, 72,
362:    77, 84, 89, 94, 98, 102, 110
363: };
364: #endif
365:
366: #if YYDEBUG || YYERROR_VERBOSE
367: /* YYTNME[SYMBOL-NUM] -- String name of the symbol SYMBOL-NUM.

```

```

368: First, the terminals, then, starting at YYNTOKENS, nonterminals. */
369: static const char *const yytname[] =
370: {
371: "$end", "error", "$undefined", "LT_SY", "CMD_MSG_SY", "DATA_MSG_SY",
372: "NAME_SY", "EQUAL_SY", "SLASH_SY", "GT_SY", "COLON_SY", "VAR_SY",
373: "LENGTH_SY", "COUNTMAX_SY", "RATEMAX_SY", "RQST_SY", "REPLY_SY", "ID_SY",
374: "FAULT_MSG_SY", "CMD_SY", "NOTI_SY", "DATATYPE", "FLOAT", "INT",
375: "STRING", "NOT_SPECIFIED_SY", "$accept", "INFO", "FAULT_MSG", "MSG_DEF",
376: "VARIABLES", "VARIABLE", "ATTRIBUTE", 0
377: };
378: #endif
379:
380: # ifdef YYPRINT
381: /* YYTOKNUM[YYLEX-NUM] -- Internal token number corresponding to
382:  token YYLEX-NUM. */
383: static const unsigned short int yytoknum[] =
384: {
385:     0, 256, 257, 258, 259, 260, 261, 262, 263, 264,
386:     265, 266, 267, 268, 269, 270, 271, 272, 273, 274,
387:     275, 276, 277, 278, 279, 280
388: };
389: # endif
390:
391: /* YYR1[YYN] -- Symbol number of symbol that rule YYN derives. */
392: static const unsigned char yyr1[] =
393: {
394:     0, 26, 27, 27, 27, 27, 27, 27, 28, 29,
395:     29, 30, 30, 31, 31, 31, 32
396: };
397:
398: /* YYR2[YYN] -- Number of symbols composing right hand side of rule YYN. */
399: static const unsigned char yyr2[] =
400: {
401:     0, 2, 15, 19, 11, 15, 11, 15, 9, 9,
402:     9, 2, 0, 13, 13, 13, 3
403: };
404:
405: /* YYDEFAC[STATE-NAME] -- Default rule to reduce with in state
406:  STATE-NAME when YYTABLE doesn't specify something else to do. Zero
407:  means the default is an error. */
408: static const unsigned char yydefact[] =

```

```

409: {
410:     0,  0,  0,  0,  0,  0,  1,  0,  0,  0,
411:     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
412:     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
413:     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
414:     0,  0,  0,  0,  0,  0,  0,  0,  0, 12,
415:    12,  0,  4,  0,  0,  6,  0,  0,  0,  0,
416:     0,  0,  0,  0,  0, 11,  9, 10,  0,  0,
417:     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
418:     0, 16,  2,  0, 12,  5,  7,  0,  0,  0,
419:     0,  0,  8,  0,  0,  0,  3,  0,  0,  0,
420:     0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
421:    15, 14, 13
422: };
423:
424: /* YYDEFGOTO[NTERM-NUM]. */
425: static const yysigned_char yydefgoto[] =
426: {
427:     -1,  2, 38, 15, 57, 65, 66
428: };
429:
430: /* YYPACT[STATE-NUM] -- Index in YYTABLE of the portion describing
431:    STATE-NUM. */
432: #define YYPACT_NINF -58
433: static const yysigned_char yypact[] =
434: {
435:     8, -7, 14, 11, 16, 17, -58, 24, 25, 27,
436:     5,  5,  5, 29, 30, 23, 31, 32, 26, 34,
437:    28, 33, 35, 19, 21, 43, 44, 45, 36, 37,
438:     5, -3, -2, 42, 48, 49, 39, 46, 51, 18,
439:    52, 38, 40, 41, 47, 54, 56, 57, 58, -58,
440:   -58, 60, -58, 50, 65, -58, 66, 12, 12, -1,
441:    53, 63, 64, 67, 68, -58, -58, -58, 61, 70,
442:    72, 62, 69, 59, 71, 74, 75, 73, 76, 77,
443:    78, -58, -58, 84, -58, -58, -58, 79, 82, 12,
444:    81, 83, -58, 85, 87, 89, -58, -4, 90, 91,
445:    93, 88, 92, 94, 86, 97, 98, 95, 96, 99,
446:   -58, -58, -58
447: };
448:
449: /* YYPGOTO[NTERM-NUM]. */

```

```

450: static const yysigned_char yypgoto[] =
451: {
452:   -58, -58, -30, -8, -50, -58, -57
453: };
454:
455: /* YYTABLE[YYPACT[STATE-NUM]]. What to do in state STATE-NUM. If
456:  positive, shift that token. If negative, reduce the rule which
457:  number is the opposite. If zero, do what YYDEFACT says.
458:  If YYTABLE_NINF, syntax error. */
459: #define YYTABLE_NINF -1
460: static const unsigned char yytable[] =
461: {
462:   58, 67, 40, 16, 17, 36, 39, 68, 3, 13,
463:   14, 1, 4, 5, 6, 37, 37, 37, 98, 99,
464:   7, 100, 35, 63, 64, 8, 9, 10, 11, 69,
465:   12, 20, 92, 23, 89, 18, 19, 25, 47, 21,
466:   22, 24, 26, 28, 27, 29, 30, 31, 32, 41,
467:   51, 0, 45, 33, 34, 42, 52, 43, 44, 46,
468:   48, 53, 49, 59, 50, 54, 55, 56, 61, 62,
469:   70, 71, 72, 0, 60, 74, 75, 73, 76, 77,
470:   0, 78, 80, 82, 83, 85, 86, 88, 87, 79,
471:   91, 93, 0, 107, 0, 81, 96, 84, 94, 97,
472:  101, 102, 90, 103, 108, 109, 0, 95, 0, 104,
473:   0, 0, 0, 105, 0, 106, 0, 0, 0, 110,
474:  111, 0, 0, 112
475: };
476:
477: static const yysigned_char yycheck[] =
478: {
479:   50, 58, 32, 11, 12, 8, 8, 8, 15, 4,
480:   5, 3, 19, 20, 0, 18, 18, 18, 22, 23,
481:   9, 25, 30, 11, 12, 9, 9, 3, 3, 59,
482:   3, 8, 89, 7, 84, 6, 6, 9, 20, 8,
483:   8, 7, 9, 24, 9, 24, 3, 3, 3, 7,
484:   9, -1, 6, 17, 17, 7, 9, 8, 19, 8,
485:   8, 7, 24, 3, 24, 9, 9, 9, 3, 3,
486:  17, 8, 8, -1, 24, 7, 15, 10, 8, 7,
487: -1, 19, 23, 9, 9, 9, 9, 3, 10, 20,
488:   8, 10, -1, 7, -1, 24, 9, 24, 15, 10,
489:  10, 10, 23, 10, 7, 7, -1, 22, -1, 21,
490: -1, -1, -1, 21, -1, 21, -1, -1, -1, 24,

```



```

491:    24, -1, -1, 24
492: };
493:
494: /* YYSTOS[STATE-NUM] -- The (internal number of the) accessing
495:    symbol of state STATE-NUM. */
496: static const unsigned char yystos[] =
497: {
498:     0,  3, 27, 15, 19, 20,  0,  9,  9,  9,
499:     3,  3,  3,  4,  5, 29, 29, 29,  6,  6,
500:     8,  8,  8,  7,  7,  9,  9,  9, 24, 24,
501:     3,  3,  3, 17, 17, 29,  8, 18, 28,  8,
502:    28,  7,  7,  8, 19,  6,  8, 20,  8, 24,
503:    24,  9,  9,  7,  9,  9,  9, 30, 30,  3,
504:    24,  3,  3, 11, 12, 31, 32, 32,  8, 28,
505:    17,  8,  8, 10,  7, 15,  8,  7, 19, 20,
506:    23, 24,  9,  9, 24,  9,  9, 10,  3, 30,
507:    23,  8, 32, 10, 15, 22,  9, 10, 22, 23,
508:    25, 10, 10, 10, 21, 21, 21,  7,  7,  7,
509:    24, 24, 24
510: };
511:
512: #if ! defined (YYSIZE_T) && defined (__SIZE_TYPE__)
513: # define YYSIZE_T __SIZE_TYPE__
514: #endif
515: #if ! defined (YYSIZE_T) && defined (size_t)
516: # define YYSIZE_T size_t
517: #endif
518: #if ! defined (YYSIZE_T)
519: # if defined (__STDC__) || defined (__cplusplus)
520: #  include <stddef.h> /* INFRINGES ON USER NAME SPACE */
521: #  define YYSIZE_T size_t
522: # endif
523: #endif
524: #if ! defined (YYSIZE_T)
525: # define YYSIZE_T unsigned int
526: #endif
527:
528: #define yyerrok      (yyerrstatus = 0)
529: #define yyclearin    (yychar = YYEMPTY)
530: #define YYEMPTY      (-2)
531: #define YYEOF        0

```

```

532:
533: #define YYACCEPT    goto yyacceptlab
534: #define YYABORT      goto yyabortlab
535: #define YYERROR      goto yyerrorlab
536:
537:
538: /* Like YYERROR except do call yyerror.  This remains here temporarily
539:  to ease the transition to the new meaning of YYERROR, for GCC.
540:  Once GCC version 2 has supplanted version 1, this can go.  */
541:
542: #define YYFAIL      goto yyerrlab
543:
544: #define YYRECOVERING() (!!yyerrstatus)
545:
546: #define YYBACKUP(Token, Value)          \
547: do                                     \
548:   if (yychar == YYEMPTY && yylen == 1) \
549:   {                                     \
550:     yychar = (Token);                  \
551:     yylval = (Value);                  \
552:     yytoken = YYTRANSLATE (yychar);    \
553:     YYPOPSTACK;                        \
554:     goto yybackup;                     \
555:   }                                     \
556: else                                  \
557:   {                                     \
558:     yyerror ("syntax error: cannot back up"); \
559:     YYERROR;                            \
560:   }                                     \
561: while (0)
562:
563: #define YYTERROR 1
564: #define YYERRCODE 256
565:
566: /* YYLLOC_DEFAULT -- Compute the default location (before the actions
567:  are run).  */
568:
569: #ifndef YYLLOC_DEFAULT
570: # define YYLLOC_DEFAULT(Current, Rhs, N)      \
571:   ((Current).first_line = (Rhs)[1].first_line, \
572:    (Current).first_column = (Rhs)[1].first_column, \

```

```

573: (Current).last_line = (Rhs)[N].last_line, \
574: (Current).last_column = (Rhs)[N].last_column)
575: #endif
576:
577: /* YYLEX -- calling `yylex' with the right arguments. */
578:
579: #ifdef YYLEX_PARAM
580: # define YYLEX yylex (YYLEX_PARAM)
581: #else
582: # define YYLEX yylex ()
583: #endif
584:
585: /* Enable debugging if requested. */
586: #if YYDEBUG
587:
588: # ifndef YYFPRINTF
589: # include <stdio.h> /* INFRINGES ON USER NAME SPACE */
590: # define YYFPRINTF fprintf
591: # endif
592:
593: # define YYDPRINTF(Args) \
594: do { \
595:   if (yydebug) \
596:     YYFPRINTF Args; \
597: } while (0)
598:
599: # define YYDSYMPRINT(Args) \
600: do { \
601:   if (yydebug) \
602:     yysymprint Args; \
603: } while (0)
604:
605: # define YYDSYMPRINTF(Title, Token, Value, Location) \
606: do { \
607:   if (yydebug) \
608:   { \
609:     YYFPRINTF (stderr, "%s ", Title); \
610:     yysymprint (stderr, \
611:                 Token, Value); \
612:     YYFPRINTF (stderr, "\n"); \
613:   } \

```

```

614: } while (0)
615:
616: /*-----.
617: | yy_stack_print -- Print the state stack from its BOTTOM up to its |
618: | TOP (included).                                         |
619: `-----*/
620:
621: #if defined (__STDC__) || defined (__cplusplus)
622: static void
623: yy_stack_print (short int *bottom, short int *top)
624: #else
625: static void
626: yy_stack_print (bottom, top)
627:   short int *bottom;
628:   short int *top;
629: #endif
630: {
631:   YYFPRINTF (stderr, "Stack now");
632:   for (/* Nothing. */; bottom <= top; ++bottom)
633:     YYFPRINTF (stderr, " %d", *bottom);
634:   YYFPRINTF (stderr, " \n");
635: }
636:
637: # define YY_STACK_PRINT(Bottom, Top)                        \
638: do {                                                         \
639:   if (yydebug)                                              \
640:     yy_stack_print ((Bottom), (Top));                      \
641: } while (0)
642:
643:
644: /*-----.
645: | Report that the YYRULE is going to be reduced. |
646: `-----*/
647:
648: #if defined (__STDC__) || defined (__cplusplus)
649: static void
650: yy_reduce_print (int yyrule)
651: #else
652: static void
653: yy_reduce_print (yyrule)
654:   int yyrule;

```

```

655: #endif
656: {
657:   int yyi;
658:   unsigned int yyno = yyline[yyrule];
659:   YYFPRINTF (stderr, "Reducing stack by rule %d (line %u), ",
660:             yyrule - 1, yyno);
661:   /* Print the symbols being reduced, and their result. */
662:   for (yyi = yyrhs[yyrule]; 0 <= yyrhs[yyi]; yyi++)
663:     YYFPRINTF (stderr, "%s ", yytname [yyrhs[yyi]]);
664:   YYFPRINTF (stderr, "-> %s \n", yytname [yyr1[yyrule]]);
665: }
666:
667: # define YY_REDUCE_PRINT(Rule)      \
668: do {                                \
669:   if (yydebug)                      \
670:     yy_reduce_print (Rule);        \
671: } while (0)
672:
673: /* Nonzero means print parse trace.  It is left uninitialized so that
674:    multiple parsers can coexist. */
675: int yydebug;
676: #else /* !YYDEBUG */
677: # define YYDPRINTF(Args)
678: # define YYDSYMPRINT(Args)
679: # define YYDSYMPRINTF(Title, Token, Value, Location)
680: # define YY_STACK_PRINT(Bottom, Top)
681: # define YY_REDUCE_PRINT(Rule)
682: #endif /* !YYDEBUG */
683:
684:
685: /* YYINITDEPTH -- initial size of the parser's stacks. */
686: #ifndef YYINITDEPTH
687: # define YYINITDEPTH 200
688: #endif
689:
690: /* YYMAXDEPTH -- maximum size the stacks can grow to (effective only
691:    if the built-in stack extension method is used).
692:
693:    Do not make this value too large; the results are undefined if
694:    SIZE_MAX < YYSTACK_BYTES (YYMAXDEPTH)
695:    evaluated with infinite-precision integer arithmetic. */

```

```

696:
697: #if defined (YYMAXDEPTH) && YYMAXDEPTH == 0
698: # undef YYMAXDEPTH
699: #endif
700:
701: #ifndef YYMAXDEPTH
702: # define YYMAXDEPTH 10000
703: #endif
704:
705:
706:
707: #if YYERROR_VERBOSE
708:
709: # ifndef yystrlen
710: #  if defined (__GLIBC__) && defined (_STRING_H)
711: #   define yystrlen strlen
712: #  else
713: /* Return the length of YYSTR. */
714: static YYSIZE_T
715: #   if defined (__STDC__) || defined (__cplusplus)
716: yystrlen (const char *yystr)
717: #   else
718: yystrlen (yystr)
719:     const char *yystr;
720: #   endif
721: {
722:   register const char *yys = yystr;
723:
724:   while (*yys++ != '\0')
725:     continue;
726:
727:   return YYS - yystr - 1;
728: }
729: # endif
730: # endif
731:
732: # ifndef yystpcpy
733: #  if defined (__GLIBC__) && defined (_STRING_H) && defined (_GNU_SOURCE)
734: #   define yystpcpy stpcpy
735: #  else
736: /* Copy YYSRC to YYDEST, returning the address of the terminating '\0' in

```

```

737: YYDEST. */
738: static char *
739: # if defined (__STDC__) || defined (__cplusplus)
740: yystpcpy (char *yydest, const char *yysrc)
741: # else
742: yystpcpy (yydest, yysrc)
743:   char *yydest;
744:   const char *yysrc;
745: # endif
746: {
747:   register char *yyd = yydest;
748:   register const char *yys = yysrc;
749:
750:   while ((*yyd++ = *yys++) != '\0')
751:     continue;
752:
753:   return yyd - 1;
754: }
755: # endif
756: # endif
757:
758: #endif /* !YYERROR_VERBOSE */
759:
760:
761:
762: #if YYDEBUG
763: /*-----
764: | Print this symbol on YYOUTPUT. |
765: `-----*/
766:
767: #if defined (__STDC__) || defined (__cplusplus)
768: static void
769: yysymprint (FILE *yyoutput, int yytype, YYSTYPE *yyvaluep)
770: #else
771: static void
772: yysymprint (yyoutput, yytype, yyvaluep)
773:   FILE *yyoutput;
774:   int yytype;
775:   YYSTYPE *yyvaluep;
776: #endif
777: {

```

```

778: /* Pacify ``unused variable" warnings. */
779: (void) yyvaluep;
780:
781: if (yytype < YYNTOKENS)
782: {
783:     YYFPRINTF (yyoutput, "token %s (", yyname[yytype]);
784: #ifdef YYPRINT
785:     YYPRINT (yyoutput, yytoknum[yytype], *yyvaluep);
786: #endif
787: }
788: else
789:     YYFPRINTF (yyoutput, "nterm %s (", yyname[yytype]);
790:
791: switch (yytype)
792: {
793:     default:
794:         break;
795: }
796: YYFPRINTF (yyoutput, ")");
797: }
798:
799: #endif /* ! YYDEBUG */
800: /*-----
801: | Release the memory associated to this symbol. |
802: `-----*/
803:
804: #if defined (__STDC__) || defined (__cplusplus)
805: static void
806: yydestruct (int yytype, YYSTYPE *yyvaluep)
807: #else
808: static void
809: yydestruct (yytype, yyvaluep)
810:     int yytype;
811:     YYSTYPE *yyvaluep;
812: #endif
813: {
814:     /* Pacify ``unused variable" warnings. */
815:     (void) yyvaluep;
816:
817:     switch (yytype)
818:     {

```



```

819:
820:     default:
821:         break;
822:     }
823: }
824:
825:
826: /* Prevent warnings from -Wmissing-prototypes. */
827:
828: #ifdef YYPARSE_PARAM
829: # if defined (__STDC__) || defined (__cplusplus)
830: int yyparse (void *YYPARSE_PARAM);
831: # else
832: int yyparse ();
833: # endif
834: #else /* ! YYPARSE_PARAM */
835: # if defined (__STDC__) || defined (__cplusplus)
836: int yyparse (void);
837: # else
838: int yyparse ();
839: # endif
840: #endif /* ! YYPARSE_PARAM */
841:
842:
843:
844: /* The lookahead symbol. */
845: int yychar;
846:
847: /* The semantic value of the lookahead symbol. */
848: YYSTYPE yylval;
849:
850: /* Number of syntax errors so far. */
851: int yynerrs;
852:
853:
854:
855: /*-----
856: | yyparse. |
857: `-----*/
858:
859: #ifdef YYPARSE_PARAM

```

```

860: # if defined (__STDC__) || defined (__cplusplus)
861: int yyparse (void *YYPARSE_PARAM)
862: # else
863: int yyparse (YYPARSE_PARAM)
864: void *YYPARSE_PARAM;
865: # endif
866: #else /* ! YYPARSE_PARAM */
867: #if defined (__STDC__) || defined (__cplusplus)
868: int
869: yyparse (void)
870: #else
871: int
872: yyparse ()
873:
874: #endif
875: #endif
876: {
877:
878: register int yystate;
879: register int yyn;
880: int yyresult;
881: /* Number of tokens to shift before error messages enabled. */
882: int yyerrstatus;
883: /* Lookahead token as an internal (translated) token number. */
884: int yytoken = 0;
885:
886: /* Three stacks and their tools:
887:  `yyss': related to states,
888:  `yyvs': related to semantic values,
889:  `yyls': related to locations.
890:
891: Refer to the stacks thru separate pointers, to allow yyoverflow
892: to reallocate them elsewhere. */
893:
894: /* The state stack. */
895: short int yyssa[YYINITDEPTH];
896: short int *yyss = yyssa;
897: register short int *yyssp;
898:
899: /* The semantic value stack. */
900: YYSTYPE yyvsa[YYINITDEPTH];

```

```

901: YYSTYPE *yyvs = yyvsa;
902: register YYSTYPE *yyvsp;
903:
904:
905:
906: #define YYPOPSTACK (yyvsp--, yyssp--)
907:
908: YYSIZE_T YYSTACKsize = YYINITDEPTH;
909:
910: /* The variables used to return semantic value and location from the
911:    action routines. */
912: YYSTYPE yyval;
913:
914:
915: /* When reducing, the number of symbols on the RHS of the reduced
916:    rule. */
917: int yrlen;
918:
919: YYDPRINTF ((stderr, "Starting parse \n"));
920:
921: yynstate = 0;
922: yynerrstatus = 0;
923: yynerrs = 0;
924: yychar = YYEMPTY; /* Cause a token to be read. */
925:
926: /* Initialize stack pointers.
927:    Waste one element of value and location stack
928:    so that they stay on the same level as the state stack.
929:    The wasted elements are never initialized. */
930:
931: yyssp = yyss;
932: yyvsp = yyvs;
933:
934:
935: goto yysetstate;
936:
937: /*-----.
938: | yynewstate -- Push a new state, which is found in yynstate. |
939: `-----*/
940: yynewstate:
941: /* In all cases, when you get here, the value and location stacks

```

```

942:   have just been pushed. so pushing a state here evens the stacks.
943:   */
944:   yyssp++;
945:
946: yysetstate:
947:   *yyssp = yystate;
948:
949:   if (yyss + yystacksize - 1 <= yyssp)
950:   {
951:       /* Get the current used size of the three stacks, in elements. */
952:       YYSIZE_T yysize = yyssp - yyss + 1;
953:
954: #ifdef yyoverflow
955:     {
956:       /* Give user a chance to reallocate the stack. Use copies of
957:          these so that the &'s don't force the real ones into
958:          memory. */
959:       YYSTYPE *yyvs1 = yyvs;
960:       short int *yyss1 = yyss;
961:
962:
963:       /* Each stack pointer address is followed by the size of the
964:          data in use in that stack, in bytes. This used to be a
965:          conditional around just the two extra args, but that might
966:          be undefined if yyoverflow is a macro. */
967:       yyoverflow ("parser stack overflow",
968:                  &yyss1, yysize * sizeof (*yyssp),
969:                  &yyvs1, yysize * sizeof (*yyvsp),
970:
971:                  &yystacksize);
972:
973:       yyss = yyss1;
974:       yyvs = yyvs1;
975:     }
976: #else /* no yyoverflow */
977: #ifndef YYSTACK_RELOCATE
978:     goto yyoverflowlab;
979: # else
980:     /* Extend the stack our own way. */
981:     if (YYMAXDEPTH <= yystacksize)
982:       goto yyoverflowlab;

```

```

983:   yystacksize *= 2;
984:   if (YYMAXDEPTH < yystacksize)
985:     yystacksize = YYMAXDEPTH;
986:
987:   {
988:     short int *yyss1 = yyss;
989:     union yyalloc *yyptr =
990:       (union yyalloc *) YYSTACK_ALLOC (YYSTACK_BYTES (ystacksize));
991:     if (! yyptr)
992:       goto yyoverflowlab;
993:     YYSTACK_RELOCATE (yyss);
994:     YYSTACK_RELOCATE (yyvs);
995:
996: # undef YYSTACK_RELOCATE
997:     if (yyss1 != yyssa)
998:       YYSTACK_FREE (yyss1);
999:   }
1000: # endif
1001: #endif /* no yyoverflow */
1002:
1003:   yyssp = yyss + yysize - 1;
1004:   yyvsp = yyvs + yysize - 1;
1005:
1006:
1007:   YYDPRINTF ((stderr, "Stack size increased to %lu \n",
1008:     (unsigned long int) yystacksize));
1009:
1010:   if (yyss + yystacksize - 1 <= yyssp)
1011:     YYABORT;
1012: }
1013:
1014: YYDPRINTF ((stderr, "Entering state %d \n", yystate));
1015:
1016: goto yybackup;
1017:
1018: /*-----,
1019: |yybackup. |
1020: `-----*/
1021: yybackup;
1022:
1023: /* Do appropriate processing given the current state. */

```

```

1024: /* Read a lookahead token if we need one and don't already have one. */
1025: /* yyresume: */
1026:
1027: /* First try to decide what to do without reference to lookahead token. */
1028:
1029: yyn = yypact[yystate];
1030: if (yyn == YYPACT_NINF)
1031:     goto yydefault;
1032:
1033: /* Not known => get a lookahead token if don't already have one. */
1034:
1035: /* YYCHAR is either YYEMPTY or YYEOF or a valid lookahead symbol. */
1036: if (yychar == YYEMPTY)
1037: {
1038:     YYDPRINTF ((stderr, "Reading a token: "));
1039:     yychar = YYLEX;
1040: }
1041:
1042: if (yychar <= YYEOF)
1043: {
1044:     yychar = yytoken = YYEOF;
1045:     YYDPRINTF ((stderr, "Now at end of input. \n"));
1046: }
1047: else
1048: {
1049:     yytoken = YYTRANSLATE (yychar);
1050:     YYDSYMPRINTF ("Next token is", yytoken, &yylval, &yyllloc);
1051: }
1052:
1053: /* If the proper action on seeing token YYTOKEN is to reduce or to
1054:    detect an error, take that action. */
1055: yyn += yytoken;
1056: if (yyn < 0 || YYLAST < yyn || yycheck[yyn] != yytoken)
1057:     goto yydefault;
1058: yyn = yytable[yyn];
1059: if (yyn <= 0)
1060: {
1061:     if (yyn == 0 || yyn == YYTABLE_NINF)
1062:         goto yyerrlab;
1063:     yyn = -yyn;
1064:     goto yyreduce;

```

```

1065: }
1066:
1067: if (yyn == YYFINAL)
1068:   YYACCEPT;
1069:
1070: /* Shift the lookahead token. */
1071: YYDPRINTF ((stderr, "Shifting token %s, ", yytname[yytoken]));
1072:
1073: /* Discard the token being shifted unless it is eof. */
1074: if (yychar != YYEOF)
1075:   yychar = YYEMPTY;
1076:
1077: *++yyvsp = yylval;
1078:
1079:
1080: /* Count tokens shifted since error; after three, turn off error
1081:    status. */
1082: if (yyerrstatus)
1083:   yyerrstatus--;
1084:
1085: yystate = yyn;
1086: goto yynewstate;
1087:
1088:
1089: /*-----
1090: | yydefault -- do the default action for the current state. |
1091: `-----*/
1092: yydefault:
1093:   yyn = yydefact[yystate];
1094:   if (yyn == 0)
1095:     goto yyerrlab;
1096:   goto yyreduce;
1097:
1098:
1099: /*-----
1100: | yyreduce -- Do a reduction. |
1101: `-----*/
1102: yyreduce:
1103:   /* yyn is the number of a rule to reduce with. */
1104:   yrlen = yyr2[yyn];
1105:

```

```

1106: /* If YYLEN is nonzero, implement the default value of the action:
1107:   `$$ = $1'.
1108:
1109:   Otherwise, the following line sets YYVAL to garbage.
1110:   This behavior is undocumented and Bison
1111:   users should not rely upon it. Assigning to YYVAL
1112:   unconditionally makes the parser a bit smaller, and it avoids a
1113:   GCC warning that YYVAL may be used uninitialized. */
1114: yyval = yyvsp[1-yylen];
1115:
1116:
1117: YY_REDUCE_PRINT (yyn);
1118: switch (yyn)
1119: {
1120:   case 2:
1121: #line 40 "msgdef.y"
1122:   {
1123:       setMessage(yyvsp[-10].node,yyvsp[-6].node,NULL);
1124:   };}
1125:   break;
1126:
1127:   case 3:
1128: #line 44 "msgdef.y"
1129:   {
1130:       setMessage(yyvsp[-14].node,yyvsp[-10].node,yyvsp[-6].node);
1131:   };}
1132:   break;
1133:
1134:   case 4:
1135: #line 48 "msgdef.y"
1136:   {
1137:       setMessage(yyvsp[-6].node,NULL,NULL);
1138:   };}
1139:   break;
1140:
1141:   case 5:
1142: #line 52 "msgdef.y"
1143:   {
1144:       setMessage(yyvsp[-10].node,NULL,yyvsp[-6].node);
1145:   };}
1146:   break;

```



```

1147:
1148: case 6:
1149: #line 56 "msgdef.y"
1150: {
1151:     setMessage(NULL,yyvsp[-6].node,NULL);
1152: };
1153: break;
1154:
1155: case 7:
1156: #line 60 "msgdef.y"
1157: {
1158:     setMessage(NULL,yyvsp[-10].node,yyvsp[-6].node);
1159: };
1160: break;
1161:
1162: case 8:
1163: #line 66 "msgdef.y"
1164: {
1165:     yyval.node = newNode(yyvsp[-5].str,FAULTMSG,yyvsp[0].integer,atoi(yyvsp[-2].str),yyvsp[-1].var);
1166:     if (yyvsp[-2].str != NULL) free(yyvsp[-2].str);
1167: };
1168: break;
1169:
1170: case 9:
1171: #line 73 "msgdef.y"
1172: {
1173:     yyval.node = newNode(yyvsp[-5].str,COMMANDMSG,yyvsp[0].integer,atoi(yyvsp[-2].str),yyvsp[-1].var);
1174:     if (yyvsp[-2].str != NULL) free(yyvsp[-2].str);
1175: };
1176: break;
1177:
1178: case 10:
1179: #line 78 "msgdef.y"
1180: {
1181:     yyval.node = newNode(yyvsp[-5].str,DATAMSG,yyvsp[0].integer,atoi(yyvsp[-2].str),yyvsp[-1].var);
1182:     if (yyvsp[-2].str != NULL) free(yyvsp[-2].str);
1183: };
1184: break;
1185:

```

```

1186: case 11:
1187: #line 85 "msgdef.y"
1188: {
1189:     yyval.var = addVar(&yyvsp[-1].var,yyvsp[0].var);
1190:     ;}
1191: break;
1192:
1193: case 12:
1194: #line 89 "msgdef.y"
1195: {
1196:     yyval.var = NULL;
1197:     ;}
1198: break;
1199:
1200: case 13:
1201: #line 95 "msgdef.y"
1202: { /*Invalid type is not specified*/
1203:     yyval.var      =      newVar(yyvsp[0].str,yyvsp[-10].integer,yyvsp[-8].integer,yyvsp[-
1204: 2].str,yyvsp[-6].real);
1205:     ;}
1206: break;
1207:
1208: case 14:
1209: #line 99 "msgdef.y"
1210: { /*Invalid type is an int*/
1211:     yyval.var = newVarInvalidIsInt(yyvsp[0].str,yyvsp[-10].integer,yyvsp[-8].integer,yyvsp[-
1212: 2].str,yyvsp[-6].real,yyvsp[-4].integer);
1213:     ;}
1214: break;
1215:
1216: case 15:
1217: #line 103 "msgdef.y"
1218: { /*Invalid type is a float*/
1219:     yyval.var      =      newVarInvalidIsFloat(yyvsp[0].str,yyvsp[-10].integer,yyvsp[-
1220: 8].integer,yyvsp[-2].str,yyvsp[-6].real,yyvsp[-4].real);
1221:     ;}
1222: break;
1223:
1224: case 16:
1225: #line 111 "msgdef.y"
1226: {
1227:     yyval.integer = atoi(yyvsp[0].str);

```

```

1225:         if (yyvsp[0].str != NULL) free(yyvsp[0].str);
1226:     ;}
1227:     break;
1228:
1229:
1230: }
1231:
1232: /* Line 1010 of yacc.c. */
1233: #line 1234 "msgdef.tab.c"
1234:
1235: yyvsp -= yrlen;
1236: yyssp -= yrlen;
1237:
1238:
1239: YY_STACK_PRINT (yys, yyssp);
1240:
1241: *++yyvsp = yyval;
1242:
1243:
1244: /* Now `shift' the result of the reduction. Determine what state
1245:    that goes to, based on the state we popped back to and the rule
1246:    number reduced by. */
1247:
1248: yyn = yyr1[yyn];
1249:
1250: yystate = yypgoto[yyn - YYNTOKENS] + *yyssp;
1251: if (0 <= yystate && yystate <= YYLAST && yycheck[yystate] == *yyssp)
1252:     yystate = yytable[yystate];
1253: else
1254:     yystate = yydefgoto[yyn - YYNTOKENS];
1255:
1256: goto yynewstate;
1257:
1258:
1259: /*-----,
1260: | yyerrlab -- here on detecting error |
1261: `-----*/
1262: yyerrlab:
1263: /* If not already recovering from an error, report this error. */
1264: if (!yyerrstatus)
1265:     {

```

```

1266:    ++yynerrs;
1267: #if YYERROR_VERBOSE
1268:     yyn = yypact[yystate];
1269:
1270:     if (YYPACT_NINF < yyn && yyn < YYLAST)
1271:     {
1272:         YYSIZE_T yysize = 0;
1273:         int yytype = YYTRANSLATE (yychar);
1274:         const char* yyprefix;
1275:         char *yymsg;
1276:         int yyx;
1277:
1278:         /* Start YYX at -YYN if negative to avoid negative indexes in
1279:            YYCHECK. */
1280:         int yyxbegin = yyn < 0 ? -yyn : 0;
1281:
1282:         /* Stay within bounds of both yycheck and yyname. */
1283:         int yychecklim = YYLAST - yyn;
1284:         int yyxend = yychecklim < YYNTOKENS ? yychecklim : YYNTOKENS;
1285:         int yycount = 0;
1286:
1287:         yyprefix = ", expecting ";
1288:         for (yyx = yyxbegin; yyx < yyxend; ++yyx)
1289:             if (yycheck[yyx + yyn] == yyx && yyx != YYTERROR)
1290:             {
1291:                 yysize += yystrlen (yyprefix) + yystrlen (yytname [yyx]);
1292:                 yycount += 1;
1293:                 if (yycount == 5)
1294:                 {
1295:                     yysize = 0;
1296:                     break;
1297:                 }
1298:             }
1299:         yysize += (sizeof ("syntax error, unexpected ")
1300:                    + yystrlen (yytname[yytype]));
1301:         yymsg = (char *) YYSTACK_ALLOC (yysize);
1302:         if (yymsg != 0)
1303:         {
1304:             char *yyp = yystpcpy (yymsg, "syntax error, unexpected ");
1305:             yyp = yystpcpy (yyp, yytname[yytype]);
1306:

```

```

1307:     if (yycount < 5)
1308:     {
1309:         yyprefix = ", expecting ";
1310:         for (yyx = yyxbegin; yyx < yyxend; ++yyx)
1311:             if (yycheck[yyx + yyn] == yyx && yyx != YYERROR)
1312:             {
1313:                 yyp = yystpcpy (yyp, yyprefix);
1314:                 yyp = yystpcpy (yyp, yytname[yyx]);
1315:                 yyprefix = " or ";
1316:             }
1317:     }
1318:     yyerror (yymsg);
1319:     YYSTACK_FREE (yymsg);
1320: }
1321: else
1322:     yyerror ("syntax error; also virtual memory exhausted");
1323: }
1324: else
1325: #endif /* YYERROR_VERBOSE */
1326: yyerror ("syntax error");
1327: }
1328:
1329:
1330:
1331: if (yyerrstatus == 3)
1332: {
1333:     /* If just tried and failed to reuse lookahead token after an
1334:     error, discard it. */
1335:
1336:     if (yychar <= YYEOF)
1337:     {
1338:         /* If at end of input, pop the error token,
1339:         then the rest of the stack, then return failure. */
1340:         if (yychar == YYEOF)
1341:             for (;;)
1342:             {
1343:                 YYPOPSTACK;
1344:                 if (yyssp == yyss)
1345:                     YYABORT;
1346:                 YYDSYMPRINTF ("Error: popping", yystos[*yyssp], yyvsp, yylsp);
1347:                 yydestruct (yystos[*yyssp], yyvsp);

```

```

1348:     }
1349: }
1350: else
1351: {
1352:     YYDSYMPPRINTF ("Error: discarding", yytoken, &yylval, &yyloc);
1353:     yydestruct (yytoken, &yylval);
1354:     yychar = YYEMPTY;
1355:
1356: }
1357: }
1358:
1359: /* Else will try to reuse lookahead token after shifting the error
1360:    token. */
1361: goto yyerrlab1;
1362:
1363:
1364: /*-----
1365: | yyerrorlab -- error raised explicitly by YYERROR. |
1366: `-----*/
1367: yyerrorlab:
1368:
1369: #ifdef __GNUC__
1370: /* Pacify GCC when the user code never invokes YYERROR and the label
1371:    yyerrorlab therefore never appears in user code. */
1372: if (0)
1373:     goto yyerrorlab;
1374: #endif
1375:
1376: yyvsp -= yylen;
1377: yyssp -= yylen;
1378: yystate = *yyssp;
1379: goto yyerrlab1;
1380:
1381:
1382: /*-----
1383: | yyerrlab1 -- common code for both syntax error and YYERROR. |
1384: `-----*/
1385: yyerrlab1:
1386: yyerrstatus = 3; /* Each real token shifted decrements this. */
1387:
1388: for (;;)

```

```

1389: {
1390:     yyn = yypact[yystate];
1391:     if (yyn != YYPACT_NINF)
1392:     {
1393:         yyn += YYTERROR;
1394:         if (0 <= yyn && yyn <= YYLAST && yyccheck[yyn] == YYTERROR)
1395:         {
1396:             yyn = yytable[yyn];
1397:             if (0 < yyn)
1398:                 break;
1399:         }
1400:     }
1401:
1402:     /* Pop the current state because it cannot handle the error token. */
1403:     if (yyssp == yyss)
1404:         YYABORT;
1405:
1406:     YYDSYMPRINTF ("Error: popping", yystos[*yyssp], yyvsp, yylsp);
1407:     yydestruct (yystos[yystate], yyvsp);
1408:     YYPOPSTACK;
1409:     yystate = *yyssp;
1410:     YY_STACK_PRINT (yyss, yyssp);
1411: }
1412:
1413: if (yyn == YYFINAL)
1414:     YYACCEPT;
1415:
1416: YYDPRINTF ((stderr, "Shifting error token, "));
1417:
1418: *++yyvsp = yylval;
1419:
1420:
1421: yystate = yyn;
1422: goto yynewstate;
1423:
1424:
1425: /*-----
1426: | yyacceptlab -- YYACCEPT comes here. |
1427: `-----*/
1428: yyacceptlab:
1429:     yyresult = 0;

```

```

1430: goto yyreturn;
1431:
1432: /*-----.
1433: | yyabortlab -- YYABORT comes here. |
1434: `-----*/
1435: yyabortlab:
1436: yyresult = 1;
1437: goto yyreturn;
1438:
1439: #ifndef yyoverflow
1440: /*-----.
1441: | yyoverflowlab -- parser overflow comes here. |
1442: `-----*/
1443: yyoverflowlab:
1444: yyerror ("parser stack overflow");
1445: yyresult = 2;
1446: /* Fall through. */
1447: #endif
1448:
1449: yyreturn:
1450: #ifndef yyoverflow
1451: if (yyss != yyssa)
1452:   YYSTACK_FREE (yyss);
1453: #endif
1454: return yyresult;
1455: }
1456:
1457:
1458: #line 116 "msgdef.y"
1459:
1460:

```


File: sdm/common/MessageManipulator/lex.MessageManipulator.c

```
1:
2: #line 3 "lex.MessageManipulator.c"
3:
4: #define YY_INT_ALIGNED short int
5:
6: /* A lexical scanner generated by flex */
7:
8: #define FLEX_SCANNER
9: #define YY_FLEX_MAJOR_VERSION 2
10: #define YY_FLEX_MINOR_VERSION 5
11: #define YY_FLEX_SUBMINOR_VERSION 31
12: #if YY_FLEX_SUBMINOR_VERSION > 0
13: #define FLEX_BETA
14: #endif
15:
16: /* First, we deal with platform-specific or compiler-specific issues. */
17:
18: /* begin standard C headers. */
19: #include <stdio.h>
20: #include <string.h>
21: #include <errno.h>
22: #include <stdlib.h>
23:
24: /* end standard C headers. */
25:
26: /* flex integer type definitions */
27:
28: #ifndef FLEXINT_H
29: #define FLEXINT_H
30:
31: /* C99 systems have <inttypes.h>. Non-C99 systems may or may not. */
32:
33: #if defined __STDC_VERSION__ && __STDC_VERSION__ >= 199901L
34: #include <inttypes.h>
35: typedef int8_t flex_int8_t;
36: typedef uint8_t flex_uint8_t;
37: typedef int16_t flex_int16_t;
38: typedef uint16_t flex_uint16_t;
39: typedef int32_t flex_int32_t;
```

```

40: typedef uint32_t flex_uint32_t;
41: #else
42: typedef signed char flex_int8_t;
43: typedef short int flex_int16_t;
44: typedef int flex_int32_t;
45: typedef unsigned char flex_uint8_t;
46: typedef unsigned short int flex_uint16_t;
47: typedef unsigned int flex_uint32_t;
48: #endif /* ! C99 */
49:
50: /* Limits of integral types. */
51: #ifndef INT8_MIN
52: #define INT8_MIN      (-128)
53: #endif
54: #ifndef INT16_MIN
55: #define INT16_MIN     (-32767-1)
56: #endif
57: #ifndef INT32_MIN
58: #define INT32_MIN     (-2147483647-1)
59: #endif
60: #ifndef INT8_MAX
61: #define INT8_MAX      (127)
62: #endif
63: #ifndef INT16_MAX
64: #define INT16_MAX     (32767)
65: #endif
66: #ifndef INT32_MAX
67: #define INT32_MAX     (2147483647)
68: #endif
69: #ifndef UINT8_MAX
70: #define UINT8_MAX     (255U)
71: #endif
72: #ifndef UINT16_MAX
73: #define UINT16_MAX    (65535U)
74: #endif
75: #ifndef UINT32_MAX
76: #define UINT32_MAX    (4294967295U)
77: #endif
78:
79: #endif /* ! FLEXINT_H */
80:

```

```

81: #ifdef __cplusplus
82:
83: /* The "const" storage-class-modifier is valid. */
84: #define YY_USE_CONST
85:
86: #else /* !__cplusplus */
87:
88: #if __STDC__
89:
90: #define YY_USE_CONST
91:
92: #endif /* __STDC__ */
93: #endif /* !__cplusplus */
94:
95: #ifdef YY_USE_CONST
96: #define yyconst const
97: #else
98: #define yyconst
99: #endif
100:
101: /* Returned upon end-of-file. */
102: #define YY_NULL 0
103:
104: /* Promotes a possibly negative, possibly signed char to an unsigned
105:  * integer for use as an array index. If the signed char is negative,
106:  * we want to instead treat it as an 8-bit unsigned char, hence the
107:  * double cast.
108:  */
109: #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
110:
111: /* Enter a start condition. This macro really ought to take a parameter,
112:  * but we do it the disgusting crufty way forced on us by the ()-less
113:  * definition of BEGIN.
114:  */
115: #define BEGIN (yy_start) = 1 + 2 *
116:
117: /* Translate the current start state into a value that can be later handed
118:  * to BEGIN to return to the state. The YYSTATE alias is for lex
119:  * compatibility.
120:  */
121: #define YY_START (((yy_start) - 1) / 2)

```

```

122: #define YYSTATE YY_START
123:
124: /* Action number for EOF rule of a given start state. */
125: #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
126:
127: /* Special action meaning "start processing a new file". */
128: #define YY_NEW_FILE MessageManipulatorrestart(MessageManipulatorin )
129:
130: #define YY_END_OF_BUFFER_CHAR 0
131:
132: /* Size of default input buffer. */
133: #ifndef YY_BUF_SIZE
134: #define YY_BUF_SIZE 16384
135: #endif
136:
137: #ifndef YY_TYPEDEF YY_BUFFER_STATE
138: #define YY_TYPEDEF YY_BUFFER_STATE
139: typedef struct yy_buffer_state *YY_BUFFER_STATE;
140: #endif
141:
142: extern int MessageManipulatorleng;
143:
144: extern FILE *MessageManipulatorin, *MessageManipulatorout;
145:
146: #define EOB_ACT_CONTINUE_SCAN 0
147: #define EOB_ACT_END_OF_FILE 1
148: #define EOB_ACT_LAST_MATCH 2
149:
150: #define YY_LESS_LINENO(n)
151:
152: /* Return all but the first "n" matched characters back to the input stream. */
153: #define yyless(n) \
154:   do \
155:     { \
156:       /* Undo effects of setting up MessageManipulatortext. */ \
157:       int yyless_macro_arg = (n); \
158:       YY_LESS_LINENO(yyless_macro_arg); \
159:       *yy_cp = (yy_hold_char); \
160:       YY_RESTORE_YY_MORE_OFFSET \
161:       (yy_c_buf_p) = yy_cp = yy_bp + yyless_macro_arg - YY_MORE_ADJ; \
162:       YY_DO_BEFORE_ACTION; /* set up MessageManipulatortext again */ \

```

```

163:     } \
164:   while ( 0 )
165:
166: #define unput(c) yyunput( c, (yytext_ptr) )
167:
168: /* The following is because we cannot portably get our hands on size_t
169:  * (without autoconf's help, which isn't available because we want
170:  * flex-generated scanners to compile on their own).
171:  */
172:
173: #ifndef YY_TYPEDEF_YY_SIZE_T
174: #define YY_TYPEDEF_YY_SIZE_T
175: typedef unsigned int yy_size_t;
176: #endif
177:
178: #ifndef YY_STRUCT_YY_BUFFER_STATE
179: #define YY_STRUCT_YY_BUFFER_STATE
180: struct yy_buffer_state
181: {
182:   FILE *yy_input_file;
183:
184:   char *yy_ch_buf;      /* input buffer */
185:   char *yy_buf_pos;     /* current position in input buffer */
186:
187:   /* Size of input buffer in bytes, not including room for EOB
188:    * characters.
189:    */
190:   yy_size_t yy_buf_size;
191:
192:   /* Number of characters read into yy_ch_buf, not including EOB
193:    * characters.
194:    */
195:   int yy_n_chars;
196:
197:   /* Whether we "own" the buffer - i.e., we know we created it,
198:    * and can realloc() it to grow it, and should free() it to
199:    * delete it.
200:    */
201:   int yy_is_our_buffer;
202:
203:   /* Whether this is an "interactive" input source; if so, and

```

```

204:  * if we're using stdio for input, then we want to use getc()
205:  * instead of fread(), to make sure we stop fetching input after
206:  * each newline.
207:  */
208:  int yy_is_interactive;
209:
210:  /* Whether we're considered to be at the beginning of a line.
211:   * If so, '^' rules will be active on the next match, otherwise
212:   * not.
213:   */
214:  int yy_at_bol;
215:
216:  int yy_bs_lineno; /**< The line count. */
217:  int yy_bs_column; /**< The column count. */
218:
219:  /* Whether to try to fill the input buffer when we reach the
220:   * end of it.
221:   */
222:  int yy_fill_buffer;
223:
224:  int yy_buffer_status;
225:
226: #define YY_BUFFER_NEW 0
227: #define YY_BUFFER_NORMAL 1
228:  /* When an EOF's been seen but there's still some text to process
229:   * then we mark the buffer as YY_EOF_PENDING, to indicate that we
230:   * shouldn't try reading from the input source any more.  We might
231:   * still have a bunch of tokens to match, though, because of
232:   * possible backing-up.
233:   *
234:   * When we actually see the EOF, we change the status to "new"
235:   * (via MessageManipulatorrestart()), so that the user can continue scanning by
236:   * just pointing MessageManipulatorin at a new input file.
237:   */
238: #define YY_BUFFER_EOF_PENDING 2
239:
240:  };
241: #endif /* !YY_STRUCT_YY_BUFFER_STATE */
242:
243: /* Stack of input buffers. */
244: static size_t yy_buffer_stack_top = 0; /**< index of top of stack. */

```

```

245: static size_t yy_buffer_stack_max = 0; /**< capacity of stack. */
246: static YY_BUFFER_STATE * yy_buffer_stack = 0; /**< Stack as an array. */
247:
248: /* We provide macros for accessing buffer states in case in the
249:  * future we want to put the buffer states in a more general
250:  * "scanner state".
251:  *
252:  * Returns the top of the stack, or NULL.
253:  */
254: #define YY_CURRENT_BUFFER ( (yy_buffer_stack) \
255:                             ? (yy_buffer_stack)[(yy_buffer_stack_top)] \
256:                             : NULL)
257:
258: /* Same as previous macro, but useful when we know that the buffer stack is not
259:  * NULL or when we need an lvalue. For internal use only.
260:  */
261: #define YY_CURRENT_BUFFER_LVALUE (yy_buffer_stack)[(yy_buffer_stack_top)]
262:
263: /* yy_hold_char holds the character lost when MessageManipulator text is formed. */
264: static char yy_hold_char;
265: static int yy_n_chars;      /* number of characters read into yy_ch_buf */
266: int MessageManipulatorleng;
267:
268: /* Points to current character in buffer. */
269: static char *yy_c_buf_p = (char *) 0;
270: static int yy_init = 1;     /* whether we need to initialize */
271: static int yy_start = 0;    /* start state number */
272:
273: /* Flag which is used to allow MessageManipulatorwrap()'s to do buffer switches
274:  * instead of setting up a fresh MessageManipulatorin. A bit of a hack ...
275:  */
276: static int yy_did_buffer_switch_on_eof;
277:
278: void MessageManipulatorrestart (FILE *input_file );
279: void MessageManipulator_switch_to_buffer (YY_BUFFER_STATE new_buffer );
280: YY_BUFFER_STATE MessageManipulator_create_buffer (FILE *file,int size );
281: void MessageManipulator_delete_buffer (YY_BUFFER_STATE b );
282: void MessageManipulator_flush_buffer (YY_BUFFER_STATE b );
283: void MessageManipulatorpush_buffer_state (YY_BUFFER_STATE new_buffer );
284: void MessageManipulatorpop_buffer_state (void );
285:

```

```

286: static void MessageManipulatorensure_buffer_stack (void );
287: static void MessageManipulator_load_buffer_state (void );
288: static void MessageManipulator_init_buffer (YY_BUFFER_STATE b,FILE *file );
289:
290: #define YY_FLUSH_BUFFER MessageManipulator_flush_buffer(YY_CURRENT_BUFFER )
291:
292: YY_BUFFER_STATE MessageManipulator_scan_buffer (char *base,yy_size_t size );
293: YY_BUFFER_STATE MessageManipulator_scan_string (yyconst char *yy_str );
294: YY_BUFFER_STATE MessageManipulator_scan_bytes (yyconst char *bytes,int len );
295:
296: void *MessageManipulatoralloc (yy_size_t );
297: void *MessageManipulatorrealloc (void *,yy_size_t );
298: void MessageManipulatorfree (void * );
299:
300: #define yy_new_buffer MessageManipulator_create_buffer
301:
302: #define yy_set_interactive(is_interactive) \
303:   { \
304:     if ( ! YY_CURRENT_BUFFER ){ \
305:       MessageManipulatorensure_buffer_stack (); \
306:       YY_CURRENT_BUFFER_LVALUE = \
307:         MessageManipulator_create_buffer(MessageManipulatorin,YY_BUF_SIZE ); \
308:     } \
309:     YY_CURRENT_BUFFER_LVALUE->yy_is_interactive = is_interactive; \
310:   }
311:
312: #define yy_set_bol(at_bol) \
313:   { \
314:     if ( ! YY_CURRENT_BUFFER ){ \
315:       MessageManipulatorensure_buffer_stack (); \
316:       YY_CURRENT_BUFFER_LVALUE = \
317:         MessageManipulator_create_buffer(MessageManipulatorin,YY_BUF_SIZE ); \
318:     } \
319:     YY_CURRENT_BUFFER_LVALUE->yy_at_bol = at_bol; \
320:   }
321:
322: #define YY_AT_BOL() (YY_CURRENT_BUFFER_LVALUE->yy_at_bol)
323:
324: /* Begin user sect3 */
325:
326: typedef unsigned char YY_CHAR;

```



```

327:
328: FILE *MessageManipulatorin = (FILE *) 0, *MessageManipulatorout = (FILE *) 0;
329:
330: typedef int yy_state_type;
331:
332: extern int MessageManipulatorlineno;
333:
334: int MessageManipulatorlineno = 1;
335:
336: extern char *MessageManipulatortext;
337: #define yytext_ptr MessageManipulatortext
338:
339: static yy_state_type yy_get_previous_state (void );
340: static yy_state_type yy_try_NUL_trans (yy_state_type current_state );
341: static int yy_get_next_buffer (void );
342: static void yy_fatal_error (yyconst char msg[] );
343:
344: /* Done after the current pattern has been matched and before the
345:  * corresponding action - sets up MessageManipulatortext.
346: */
347: #define YY_DO_BEFORE_ACTION \
348:   (yytext_ptr) = yy_bp; \
349:   MessageManipulatorleng = (size_t) (yy_cp - yy_bp); \
350:   (yy_hold_char) = *yy_cp; \
351:   *yy_cp = '\0'; \
352:   (yy_c_buf_p) = yy_cp;
353:
354: #define YY_NUM_RULES 25
355: #define YY_END_OF_BUFFER 26
356: /* This struct is not used in this scanner,
357:  but its presence is necessary. */
358: struct yy_trans_info
359: {
360:   flex_int32_t yy_verify;
361:   flex_int32_t yy_nxt;
362: };
363: static yyconst flex_int16_t yy_accept[145] =
364: { 0,
365:   23, 23, 26, 24, 23, 23, 23, 24, 24, 3,
366:   22, 5, 1, 2, 4, 24, 24, 24, 24, 24,
367:   24, 24, 24, 24, 24, 24, 24, 24, 23, 23,

```

```

368: 23, 0, 20, 21, 0, 22, 0, 0, 0, 0,
369: 0, 0, 0, 18, 0, 0, 0, 0, 14, 0,
370: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
371: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
372: 0, 0, 0, 0, 0, 0, 0, 0, 15, 0,
373: 0, 0, 0, 0, 0, 19, 0, 0, 0, 0,
374: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
375:
376: 0, 0, 0, 17, 0, 13, 0, 6, 0, 0,
377: 11, 0, 0, 0, 0, 8, 0, 16, 0, 0,
378: 0, 0, 0, 7, 0, 0, 0, 0, 0, 0,
379: 0, 12, 0, 0, 0, 0, 0, 0, 10, 0,
380: 0, 0, 9, 0
381: };
382:
383: static yyconst flex_int32_t yy_ec[256] =
384: { 0,
385: 1, 1, 1, 1, 1, 1, 1, 1, 2, 3,
386: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
387: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
388: 1, 4, 1, 5, 1, 1, 1, 1, 1, 1,
389: 1, 1, 1, 1, 1, 6, 7, 8, 9, 10,
390: 11, 12, 13, 14, 13, 15, 13, 16, 1, 17,
391: 18, 19, 1, 1, 20, 1, 21, 22, 23, 24,
392: 25, 1, 26, 1, 1, 27, 28, 29, 30, 31,
393: 1, 32, 33, 34, 35, 36, 1, 1, 37, 1,
394: 1, 1, 1, 1, 38, 1, 39, 40, 41, 42,
395:
396: 43, 44, 45, 46, 47, 1, 1, 48, 49, 50,
397: 51, 31, 52, 53, 54, 55, 56, 1, 1, 1,
398: 37, 1, 1, 1, 1, 1, 1, 1, 1, 1,
399: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
400: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
401: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
402: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
403: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
404: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
405: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
406:
407: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
408: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```

```

409:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
410:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
411:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
412:    1, 1, 1, 1, 1
413: };
414:
415: static yyconst flex_int32_t yy_meta[57] =
416: { 0,
417:    1, 1, 1, 1, 1, 2, 1, 2, 2, 2,
418:    2, 2, 2, 2, 2, 1, 1, 1, 1, 1,
419:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
420:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
421:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
422:    1, 1, 1, 1, 1, 1
423: };
424:
425: static yyconst flex_int16_t yy_base[147] =
426: { 0,
427:    0, 0, 245, 246, 242, 240, 238, 236, 49, 246,
428:    234, 246, 246, 246, 246, 35, 46, 40, 210, 36,
429:    195, 211, 197, 38, 51, 193, 191, 194, 230, 228,
430:    226, 224, 246, 83, 91, 222, 60, 79, 74, 76,
431:    197, 170, 191, 246, 169, 171, 193, 168, 246, 170,
432:    170, 83, 84, 95, 96, 198, 169, 64, 169, 159,
433:    180, 166, 167, 168, 97, 98, 172, 42, 175, 153,
434:    192, 192, 195, 160, 160, 73, 163, 146, 246, 90,
435:    91, 89, 146, 63, 171, 246, 151, 143, 181, 181,
436:    184, 153, 146, 101, 102, 102, 146, 180, 177, 134,
437:
438:    146, 131, 137, 246, 146, 110, 147, 246, 124, 129,
439:    246, 124, 98, 111, 103, 246, 109, 246, 106, 118,
440:    125, 80, 121, 246, 84, 63, 75, 121, 33, 121,
441:    119, 246, 133, 129, 127, 40, 134, 129, 246, 136,
442:    131, 138, 246, 246, 181, 74
443: };
444:
445: static yyconst flex_int16_t yy_def[147] =
446: { 0,
447:    144, 1, 144, 144, 144, 144, 144, 145, 144, 144,
448:    146, 144, 144, 144, 144, 144, 144, 144, 144, 144,
449:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,

```

```

450: 144, 145, 144, 144, 144, 146, 144, 144, 144, 144,
451: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
452: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
453: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
454: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
455: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
456: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
457:
458: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
459: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
460: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
461: 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
462: 144, 144, 144, 0, 144, 144
463: };
464:
465: static yyconst flex_int16_t yy_nxt[303] =
466: { 0,
467: 4, 5, 6, 7, 8, 9, 10, 11, 11, 11,
468: 11, 11, 11, 11, 11, 12, 13, 14, 15, 4,
469: 16, 17, 4, 18, 4, 19, 4, 4, 20, 4,
470: 4, 21, 4, 4, 22, 23, 4, 4, 4, 4,
471: 24, 25, 4, 4, 4, 4, 26, 27, 4, 28,
472: 4, 4, 4, 4, 4, 4, 34, 34, 34, 34,
473: 34, 34, 34, 34, 37, 39, 41, 37, 44, 83,
474: 39, 71, 72, 98, 73, 36, 99, 138, 42, 82,
475: 89, 90, 132, 91, 40, 38, 45, 52, 37, 39,
476: 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34,
477:
478: 34, 34, 34, 34, 34, 34, 52, 54, 52, 54,
479: 65, 65, 130, 129, 67, 67, 80, 80, 94, 94,
480: 96, 128, 105, 105, 107, 119, 126, 53, 54, 121,
481: 55, 65, 66, 67, 68, 80, 81, 114, 123, 94,
482: 95, 96, 105, 106, 107, 127, 119, 113, 131, 133,
483: 121, 134, 135, 136, 137, 141, 139, 140, 142, 123,
484: 143, 125, 124, 122, 120, 127, 118, 117, 116, 131,
485: 133, 135, 134, 136, 141, 137, 139, 115, 140, 142,
486: 143, 32, 32, 113, 112, 111, 110, 109, 86, 86,
487: 108, 104, 103, 86, 86, 86, 102, 101, 100, 97,
488:
489: 93, 92, 88, 87, 86, 86, 86, 85, 84, 82,
490: 79, 78, 77, 76, 75, 74, 70, 69, 64, 63,

```

```

491:    62, 61, 60, 59, 58, 57, 56, 35, 33, 31,
492:    30, 29, 51, 50, 49, 48, 47, 46, 43, 35,
493:    33, 31, 30, 29, 144, 3, 144, 144, 144, 144,
494:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
495:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
496:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
497:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
498:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
499:
500:    144, 144
501:    };
502:
503: static yyconst flex_int16_t yy_chk[303] =
504:    { 0,
505:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
506:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
507:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
508:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
509:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
510:      1, 1, 1, 1, 1, 1, 9, 9, 9, 9,
511:      9, 9, 9, 9, 16, 17, 18, 24, 20, 68,
512:      25, 58, 58, 84, 58, 146, 84, 136, 18, 68,
513:      76, 76, 129, 76, 17, 16, 20, 37, 24, 25,
514:      34, 34, 34, 34, 34, 34, 34, 34, 35, 35,
515:
516:      35, 35, 35, 35, 35, 35, 38, 39, 37, 40,
517:      52, 53, 127, 126, 54, 55, 65, 66, 80, 81,
518:      82, 125, 94, 95, 96, 113, 122, 38, 39, 115,
519:      40, 52, 53, 54, 55, 65, 66, 106, 119, 80,
520:      81, 82, 94, 95, 96, 123, 113, 106, 128, 130,
521:      115, 131, 133, 134, 135, 140, 137, 138, 141, 119,
522:      142, 121, 120, 117, 114, 123, 112, 110, 109, 128,
523:      130, 133, 131, 134, 140, 135, 137, 107, 138, 141,
524:      142, 145, 145, 105, 103, 102, 101, 100, 99, 98,
525:      97, 93, 92, 91, 90, 89, 88, 87, 85, 83,
526:
527:      78, 77, 75, 74, 73, 72, 71, 70, 69, 67,
528:      64, 63, 62, 61, 60, 59, 57, 56, 51, 50,
529:      48, 47, 46, 45, 43, 42, 41, 36, 32, 31,
530:      30, 29, 28, 27, 26, 23, 22, 21, 19, 11,
531:      8, 7, 6, 5, 3, 144, 144, 144, 144, 144,

```

```

532:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
533:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
534:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
535:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
536:    144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
537:
538:    144, 144
539:    };
540:
541: static yy_state_type yy_last_accepting_state;
542: static char *yy_last_accepting_cpos;
543:
544: extern int MessageManipulator_flex_debug;
545: int MessageManipulator_flex_debug = 0;
546:
547: /* The intent behind this definition is that it'll catch
548:  * any uses of REJECT which flex missed.
549:  */
550: #define REJECT reject_used_but_not_detected
551: #define yymore() yymore_used_but_not_detected
552: #define YY_MORE_ADJ 0
553: #define YY_RESTORE_YY_MORE_OFFSET
554: char *MessageManipulator_text;
555: #line 1 "msgdef.l"
556: #line 2 "msgdef.l"
557: #include <stdio.h>
558: #include <stdlib.h>
559: #include <string.h>
560: #include "msgdef.tab.h"
561: #line 562 "lex.MessageManipulator.c"
562:
563: #define INITIAL 0
564:
565: #ifndef YY_NO_UNISTD_H
566: /* Special case for "unistd.h", since it is non-ANSI. We include it way
567:  * down here because we want the user's section 1 to have been scanned first.
568:  * The user has a chance to override it with an option.
569:  */
570: #include <unistd.h>
571: #endif
572:

```

```

573: #ifndef YY_EXTRA_TYPE
574: #define YY_EXTRA_TYPE void *
575: #endif
576:
577: /* Macros after this point can all be overridden by user definitions in
578:  * section 1.
579:  */
580:
581: #ifndef YY_SKIP_YYWRAP
582: #ifdef __cplusplus
583: extern "C" int MessageManipulatorwrap (void );
584: #else
585: extern int MessageManipulatorwrap (void );
586: #endif
587: #endif
588:
589: #ifndef yytext_ptr
590: static void yy_flex_strncpy (char *,yyconst char *,int );
591: #endif
592:
593: #ifdef YY_NEED_STRLEN
594: static int yy_flex_strlen (yyconst char * );
595: #endif
596:
597: #ifndef YY_NO_INPUT
598:
599: #ifdef __cplusplus
600: static int yyinput (void );
601: #else
602: static int input (void );
603: #endif
604:
605: #endif
606:
607: /* Amount of stuff to slurp up with each read. */
608: #ifndef YY_READ_BUF_SIZE
609: #define YY_READ_BUF_SIZE 8192
610: #endif
611:
612: /* Copy whatever the last rule matched to the standard output. */
613: #ifndef ECHO

```

```

614: /* This used to be an fputs(), but since the string might contain NUL's,
615:  * we now use fwrite().
616:  */
617: #define ECHO (void) fwrite( MessageManipulatortext, MessageManipulatorleng, 1,
MessageManipulatorout )
618: #endif
619:
620: /* Gets input and stuffs it into "buf". number of characters read, or YY_NULL,
621:  * is returned in "result".
622:  */
623: #ifndef YY_INPUT
624: #define YY_INPUT(buf,result,max_size) \
625:   if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
626:   { \
627:     int c = '*'; \
628:     size_t n; \
629:     for ( n = 0; n < max_size && \
630:           (c = getc( MessageManipulatorin )) != EOF && c != '\n'; ++n ) \
631:       buf[n] = (char) c; \
632:     if ( c == '\n' ) \
633:       buf[n++] = (char) c; \
634:     if ( c == EOF && ferror( MessageManipulatorin ) ) \
635:       YY_FATAL_ERROR( "input in flex scanner failed" ); \
636:     result = n; \
637:   } \
638:   else \
639:   { \
640:     errno=0; \
641:     while ( (result = fread(buf, 1, max_size, MessageManipulatorin))==0 &&
ferror(MessageManipulatorin)) \
642:       { \
643:         if( errno != EINTR) \
644:           { \
645:             YY_FATAL_ERROR( "input in flex scanner failed" ); \
646:             break; \
647:           } \
648:           errno=0; \
649:           clearerr(MessageManipulatorin); \
650:       } \
651:   } \
652:   \

```



```

653:
654: #endif
655:
656: /* No semi-colon after return; correct usage is to write "yyterminate();" -
657: * we don't want an extra ';' after the "return" because that will cause
658: * some compilers to complain about unreachable statements.
659: */
660: #ifndef yyterminate
661: #define yyterminate() return YY_NULL
662: #endif
663:
664: /* Number of entries by which start-condition stack grows. */
665: #ifndef YY_START_STACK_INCR
666: #define YY_START_STACK_INCR 25
667: #endif
668:
669: /* Report a fatal error. */
670: #ifndef YY_FATAL_ERROR
671: #define YY_FATAL_ERROR(msg) yy_fatal_error( msg )
672: #endif
673:
674: /* end tables serialization structures and prototypes */
675:
676: /* Default declaration of generated scanner - a define so the user can
677: * easily add parameters.
678: */
679: #ifndef YY_DECL
680: #define YY_DECL_IS_OURS 1
681:
682: extern int MessageManipulatorlex (void);
683:
684: #define YY_DECL int MessageManipulatorlex (void)
685: #endif /* !YY_DECL */
686:
687: /* Code executed at the beginning of each rule, after MessageManipulatorlex and
688: * MessageManipulatorleng
689: * have been set up.
690: */
691: #ifndef YY_USER_ACTION
692: #define YY_USER_ACTION

```

```

693:
694: /* Code executed at the end of each rule. */
695: #ifndef YY_BREAK
696: #define YY_BREAK break;
697: #endif
698:
699: #define YY_RULE_SETUP \
700:     YY_USER_ACTION
701:
702: /** The main scanner function which does all the work.
703:  */
704: YY_DECL
705: {
706:     register yy_state_type yy_current_state;
707:     register char *yy_cp, *yy_bp;
708:     register int yy_act;
709:
710: #line 10 "msgdef.l"
711:
712:
713: #line 714 "lex.MessageManipulator.c"
714:
715:     if ( (yy_init) )
716:     {
717:         (yy_init) = 0;
718:
719: #ifdef YY_USER_INIT
720:         YY_USER_INIT;
721: #endif
722:
723:         if ( ! (yy_start) )
724:             (yy_start) = 1; /* first start state */
725:
726:         if ( ! MessageManipulatorin )
727:             MessageManipulatorin = stdin;
728:
729:         if ( ! MessageManipulatorout )
730:             MessageManipulatorout = stdout;
731:
732:         if ( ! YY_CURRENT_BUFFER ) {
733:             MessageManipulatorsure_buffer_stack ();

```

```

734:         YY_CURRENT_BUFFER_LVALUE =
735:             MessageManipulator_create_buffer(MessageManipulatorin,YY_BUF_SIZE );
736:     }
737:
738:     MessageManipulator_load_buffer_state( );
739: }
740:
741: while ( 1 )      /* loops until end-of-file is reached */
742: {
743:     yy_cp = (yy_c_buf_p);
744:
745:     /* Support of MessageManipulator text. */
746:     *yy_cp = (yy_hold_char);
747:
748:     /* yy_bp points to the position in yy_ch_buf of the start of
749:      * the current run.
750:      */
751:     yy_bp = yy_cp;
752:
753:     yy_current_state = (yy_start);
754: yy_match:
755:     do
756:     {
757:         register YY_CHAR yy_c = yy_ec[YY_SC_TO_UI(*yy_cp)];
758:         if ( yy_accept[yy_current_state] )
759:         {
760:             (yy_last_accepting_state) = yy_current_state;
761:             (yy_last_accepting_cpos) = yy_cp;
762:         }
763:         while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
764:         {
765:             yy_current_state = (int) yy_def[yy_current_state];
766:             if ( yy_current_state >= 145 )
767:                 yy_c = yy_meta[(unsigned int) yy_c];
768:         }
769:         yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
770:         ++yy_cp;
771:     }
772:     while ( yy_current_state != 144 );
773:     yy_cp = (yy_last_accepting_cpos);
774:     yy_current_state = (yy_last_accepting_state);

```

```

775:
776: yy_find_action:
777:     yy_act = yy_accept[yy_current_state];
778:
779:     YY_DO_BEFORE_ACTION;
780:
781: do_action: /* This label is used only to access EOF actions. */
782:
783:     switch ( yy_act )
784:     { /* beginning of action switch */
785:         case 0: /* must back up */
786:             /* undo the effects of YY_DO_BEFORE_ACTION */
787:             *yy_cp = (yy_hold_char);
788:             yy_cp = (yy_last_accepting_cpos);
789:             yy_current_state = (yy_last_accepting_state);
790:             goto yy_find_action;
791:
792: case 1:
793: YY_RULE_SETUP
794: #line 12 "msgdef.l"
795: {return LT_SY;}
796:     YY_BREAK
797: case 2:
798: YY_RULE_SETUP
799: #line 13 "msgdef.l"
800: {return EQUAL_SY;}
801:     YY_BREAK
802: case 3:
803: YY_RULE_SETUP
804: #line 14 "msgdef.l"
805: {return SLASH_SY;}
806:     YY_BREAK
807: case 4:
808: YY_RULE_SETUP
809: #line 15 "msgdef.l"
810: {return GT_SY;}
811:     YY_BREAK
812: case 5:
813: YY_RULE_SETUP
814: #line 16 "msgdef.l"
815: {return COLON_SY;}

```

816: YY_BREAK
817: case 6:
818: YY_RULE_SETUP
819: #line 18 "msgdef.l"
820: {return DATA_MSG_SY;}
821: YY_BREAK
822: case 7:
823: YY_RULE_SETUP
824: #line 19 "msgdef.l"
825: {return CMD_MSG_SY;}
826: YY_BREAK
827: case 8:
828: YY_RULE_SETUP
829: #line 20 "msgdef.l"
830: {return FAULT_MSG_SY;}
831: YY_BREAK
832: case 9:
833: YY_RULE_SETUP
834: #line 22 "msgdef.l"
835: {return REPLY_SY;}
836: YY_BREAK
837: case 10:
838: YY_RULE_SETUP
839: #line 23 "msgdef.l"
840: {return REPLY_SY;}
841: YY_BREAK
842: case 11:
843: YY_RULE_SETUP
844: #line 24 "msgdef.l"
845: {return RQST_SY;}
846: YY_BREAK
847: case 12:
848: YY_RULE_SETUP
849: #line 25 "msgdef.l"
850: {return NOTI_SY;}
851: YY_BREAK
852: case 13:
853: YY_RULE_SETUP
854: #line 26 "msgdef.l"
855: {return CMD_SY;}
856: YY_BREAK

```

857: case 14:
858: YY_RULE_SETUP
859: #line 28 "msgdef.l"
860: {return ID_SY;}
861:   YY_BREAK
862: case 15:
863: YY_RULE_SETUP
864: #line 30 "msgdef.l"
865: {return NAME_SY;}
866:   YY_BREAK
867: case 16:
868: YY_RULE_SETUP
869: #line 31 "msgdef.l"
870: {return VAR_SY;}
871:   YY_BREAK
872: case 17:
873: YY_RULE_SETUP
874: #line 32 "msgdef.l"
875: {return LENGTH_SY;}
876:   YY_BREAK
877: case 18:
878: YY_RULE_SETUP
879: #line 33 "msgdef.l"
880: {return NOT_SPECIFIED_SY;}
881:   YY_BREAK
882: case 19:
883: YY_RULE_SETUP
884: #line 35 "msgdef.l"
885: { MessageManipulatorlval.str = strdup(MessageManipulatorlval.text); return DATATYPE; }
886:   YY_BREAK
887: case 20:
888: /* rule 20 can match eol */
889: YY_RULE_SETUP
890: #line 37 "msgdef.l"
891:   {MessageManipulatorlval.str = strdup(MessageManipulatorlval.text+1,
strlen(MessageManipulatorlval.text)-2); return STRING;}
892:   YY_BREAK
893: case 21:
894: YY_RULE_SETUP
895: #line 38 "msgdef.l"
896: {MessageManipulatorlval.real = atof(MessageManipulatorlval.text); return FLOAT;}

```

```

897: YY_BREAK
898: case 22:
899: YY_RULE_SETUP
900: #line 39 "msgdef.l"
901: {MessageManipulatorlval.integer = atoi(MessageManipulatorlval.text); return INT;}
902: YY_BREAK
903: case 23:
904: /* rule 23 can match eol */
905: YY_RULE_SETUP
906: #line 42 "msgdef.l"
907: { /* ignore whitespace */ }
908: YY_BREAK
909: case 24:
910: YY_RULE_SETUP
911: #line 43 "msgdef.l"
912: { printf ("Invalid token \n"); }
913: YY_BREAK
914: case 25:
915: YY_RULE_SETUP
916: #line 45 "msgdef.l"
917: ECHO;
918: YY_BREAK
919: #line 920 "lex.MessageManipulator.c"
920: case YY_STATE_EOF(INITIAL):
921: yyterminate();
922:
923: case YY_END_OF_BUFFER:
924: {
925: /* Amount of text matched not including the EOB char. */
926: int yy_amount_of_matched_text = (int) (yy_cp - (yytext_ptr)) - 1;
927:
928: /* Undo the effects of YY_DO_BEFORE_ACTION. */
929: *yy_cp = (yy_hold_char);
930: YY_RESTORE_YY_MORE_OFFSET
931:
932: if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_NEW )
933: {
934: /* We're scanning a new file or input source. It's
935:  * possible that this happened because the user
936:  * just pointed MessageManipulatorin at a new source and called
937:  * MessageManipulatorlex(). If so, then we have to assure

```

```

938:      * consistency between YY_CURRENT_BUFFER and our
939:      * globals. Here is the right place to do so, because
940:      * this is the first action (other than possibly a
941:      * back-up) that will match for the new input source.
942:      */
943:      (yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
944:      YY_CURRENT_BUFFER_LVALUE->yy_input_file = MessageManipulatorin;
945:      YY_CURRENT_BUFFER_LVALUE->yy_buffer_status = YY_BUFFER_NORMAL;
946:  }
947:
948:  /* Note that here we test for yy_c_buf_p "<=" to the position
949:   * of the first EOB in the buffer, since yy_c_buf_p will
950:   * already have been incremented past the NUL character
951:   * (since all states make transitions on EOB to the
952:   * end-of-buffer state). Contrast this with the test
953:   * in input().
954:   */
955:  if ( (yy_c_buf_p) <= &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] )
956:      { /* This was really a NUL. */
957:          yy_state_type yy_next_state;
958:
959:          (yy_c_buf_p) = (yytext_ptr) + yy_amount_of_matched_text;
960:
961:          yy_current_state = yy_get_previous_state( );
962:
963:          /* Okay, we're now positioned to make the NUL
964:           * transition. We couldn't have
965:           * yy_get_previous_state() go ahead and do it
966:           * for us because it doesn't know how to deal
967:           * with the possibility of jamming (and we don't
968:           * want to build jamming into it because then it
969:           * will run more slowly).
970:           */
971:
972:          yy_next_state = yy_try_NUL_trans( yy_current_state );
973:
974:          yy_bp = (yytext_ptr) + YY_MORE_ADJ;
975:
976:          if ( yy_next_state )
977:              {
978:                  /* Consume the NUL. */

```



```

979:         yy_cp = ++(yy_c_buf_p);
980:         yy_current_state = yy_next_state;
981:         goto yy_match;
982:     }
983:
984:     else
985:     {
986:         yy_cp = (yy_last_accepting_cpos);
987:         yy_current_state = (yy_last_accepting_state);
988:         goto yy_find_action;
989:     }
990: }
991:
992: else switch ( yy_get_next_buffer( ) )
993: {
994:     case EOB_ACT_END_OF_FILE:
995:     {
996:         (yy_did_buffer_switch_on_eof) = 0;
997:
998:         if ( MessageManipulatorwrap( ) )
999:         {
1000:             /* Note: because we've taken care in
1001:              * yy_get_next_buffer() to have set up
1002:              * MessageManipulator text, we can now set up
1003:              * yy_c_buf_p so that if some total
1004:              * hoser (like flex itself) wants to
1005:              * call the scanner after we return the
1006:              * YY_NULL, it'll still work - another
1007:              * YY_NULL will get returned.
1008:              */
1009:             (yy_c_buf_p) = (yytext_ptr) + YY_MORE_ADJ;
1010:
1011:             yy_act = YY_STATE_EOF(YY_START);
1012:             goto do_action;
1013:         }
1014:
1015:     else
1016:     {
1017:         if ( ! (yy_did_buffer_switch_on_eof) )
1018:             YY_NEW_FILE;
1019:     }

```

```

1020:         break;
1021:     }
1022:
1023:     case EOB_ACT_CONTINUE_SCAN:
1024:         (yy_c_buf_p) =
1025:             (yytext_ptr) + yy_amount_of_matched_text;
1026:
1027:         yy_current_state = yy_get_previous_state( );
1028:
1029:         yy_cp = (yy_c_buf_p);
1030:         yy_bp = (yytext_ptr) + YY_MORE_ADJ;
1031:         goto yy_match;
1032:
1033:     case EOB_ACT_LAST_MATCH:
1034:         (yy_c_buf_p) =
1035:             &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)];
1036:
1037:         yy_current_state = yy_get_previous_state( );
1038:
1039:         yy_cp = (yy_c_buf_p);
1040:         yy_bp = (yytext_ptr) + YY_MORE_ADJ;
1041:         goto yy_find_action;
1042:     }
1043:     break;
1044: }
1045:
1046: default:
1047:     YY_FATAL_ERROR(
1048:         "fatal flex scanner internal error--no action found" );
1049: } /* end of action switch */
1050: } /* end of scanning one token */
1051: } /* end of MessageManipulatorlex */
1052:
1053: /* yy_get_next_buffer - try to read in a new buffer
1054:  *
1055:  * Returns a code representing an action:
1056:  *   EOB_ACT_LAST_MATCH -
1057:  *   EOB_ACT_CONTINUE_SCAN - continue scanning from current position
1058:  *   EOB_ACT_END_OF_FILE - end of file
1059:  */
1060: static int yy_get_next_buffer (void)

```

```

1061: {
1062:     register char *dest = YY_CURRENT_BUFFER_LVALUE->yy_ch_buf;
1063:     register char *source = (yytext_ptr);
1064:     register int number_to_move, i;
1065:     int ret_val;
1066:
1067:     if ( (yy_c_buf_p) > &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] )
1068:         YY_FATAL_ERROR(
1069:             "fatal flex scanner internal error--end of buffer missed" );
1070:
1071:     if ( YY_CURRENT_BUFFER_LVALUE->yy_fill_buffer == 0 )
1072:         { /* Don't try to fill the buffer, so this is an EOF. */
1073:             if ( (yy_c_buf_p) - (yytext_ptr) - YY_MORE_ADJ == 1 )
1074:                 {
1075:                     /* We matched a single character, the EOB, so
1076:                      * treat this as a final EOF.
1077:                      */
1078:                     return EOB_ACT_END_OF_FILE;
1079:                 }
1080:
1081:             else
1082:                 {
1083:                     /* We matched some text prior to the EOB, first
1084:                      * process it.
1085:                      */
1086:                     return EOB_ACT_LAST_MATCH;
1087:                 }
1088:         }
1089:
1090:     /* Try to read more data. */
1091:
1092:     /* First move last chars to start of buffer. */
1093:     number_to_move = (int) ((yy_c_buf_p) - (yytext_ptr)) - 1;
1094:
1095:     for ( i = 0; i < number_to_move; ++i )
1096:         *(dest++) = *(source++);
1097:
1098:     if ( YY_CURRENT_BUFFER_LVALUE->yy_buffer_status == YY_BUFFER_EOF_PENDING
1099:         )
1100:         /* don't do the read, it's not guaranteed to return an EOF,
1101:          * just force an EOF

```

```

1101:    */
1102:    YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars) = 0;
1103:
1104:  else
1105:    {
1106:      size_t num_to_read =
1107:        YY_CURRENT_BUFFER_LVALUE->yy_buf_size - number_to_move - 1;
1108:
1109:      while ( num_to_read <= 0 )
1110:        { /* Not enough room in the buffer - grow it. */
1111:
1112:          /* just a shorter name for the current buffer */
1113:          YY_BUFFER_STATE b = YY_CURRENT_BUFFER;
1114:
1115:          int yy_c_buf_p_offset =
1116:            (int) ((yy_c_buf_p) - b->yy_ch_buf);
1117:
1118:          if ( b->yy_is_our_buffer )
1119:            {
1120:              int new_size = b->yy_buf_size * 2;
1121:
1122:              if ( new_size <= 0 )
1123:                b->yy_buf_size += b->yy_buf_size / 8;
1124:              else
1125:                b->yy_buf_size *= 2;
1126:
1127:              b->yy_ch_buf = (char *)
1128:                /* Include room in for 2 EOB chars. */
1129:                MessageManipulatorrealloc((void *) b->yy_ch_buf,b->yy_buf_size + 2 );
1130:            }
1131:          else
1132:            /* Can't grow it, we don't own it. */
1133:            b->yy_ch_buf = 0;
1134:
1135:          if ( ! b->yy_ch_buf )
1136:            YY_FATAL_ERROR(
1137:              "fatal error - scanner input buffer overflow" );
1138:
1139:          (yy_c_buf_p) = &b->yy_ch_buf[yy_c_buf_p_offset];
1140:
1141:          num_to_read = YY_CURRENT_BUFFER_LVALUE->yy_buf_size -

```

```

1142:             number_to_move - 1;
1143:
1144:         }
1145:
1146:     if ( num_to_read > YY_READ_BUF_SIZE )
1147:         num_to_read = YY_READ_BUF_SIZE;
1148:
1149:     /* Read in more data. */
1150:     YY_INPUT( (&YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[number_to_move]),
1151:         (yy_n_chars), num_to_read );
1152:
1153:     YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
1154: }
1155:
1156: if ( (yy_n_chars) == 0 )
1157: {
1158:     if ( number_to_move == YY_MORE_ADJ )
1159:     {
1160:         ret_val = EOB_ACT_END_OF_FILE;
1161:         MessageManipulatorrestart(MessageManipulatorin );
1162:     }
1163:
1164:     else
1165:     {
1166:         ret_val = EOB_ACT_LAST_MATCH;
1167:         YY_CURRENT_BUFFER_LVALUE->yy_buffer_status =
1168:             YY_BUFFER_EOF_PENDING;
1169:     }
1170: }
1171:
1172: else
1173:     ret_val = EOB_ACT_CONTINUE_SCAN;
1174:
1175: (yy_n_chars) += number_to_move;
1176: YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] =
YY_END_OF_BUFFER_CHAR;
1177: YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars) + 1] =
YY_END_OF_BUFFER_CHAR;
1178:
1179: (yytext_ptr) = &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[0];
1180:

```

```

1181: return ret_val;
1182: }
1183:
1184: /* yy_get_previous_state - get the state just before the EOB char was reached */
1185:
1186: static yy_state_type yy_get_previous_state (void)
1187: {
1188: register yy_state_type yy_current_state;
1189: register char *yy_cp;
1190:
1191: yy_current_state = (yy_start);
1192:
1193: for ( yy_cp = (yytext_ptr) + YY_MORE_ADJ; yy_cp < (yy_c_buf_p); ++yy_cp )
1194: {
1195: register YY_CHAR yy_c = (*yy_cp ? yy_ec[YY_SC_TO_UI(*yy_cp)] : 1);
1196: if ( yy_accept[yy_current_state] )
1197: {
1198: (yy_last_accepting_state) = yy_current_state;
1199: (yy_last_accepting_cpos) = yy_cp;
1200: }
1201: while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
1202: {
1203: yy_current_state = (int) yy_def[yy_current_state];
1204: if ( yy_current_state >= 145 )
1205: yy_c = yy_meta[(unsigned int) yy_c];
1206: }
1207: yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
1208: }
1209:
1210: return yy_current_state;
1211: }
1212:
1213: /* yy_try_NUL_trans - try to make a transition on the NUL character
1214: *
1215: * synopsis
1216: * next_state = yy_try_NUL_trans( current_state );
1217: */
1218: static yy_state_type yy_try_NUL_trans (yy_state_type yy_current_state )
1219: {
1220: register int yy_is_jam;
1221: register char *yy_cp = (yy_c_buf_p);

```

```

1222:
1223: register YY_CHAR yy_c = 1;
1224: if ( yy_accept[yy_current_state] )
1225:     {
1226:         (yy_last_accepting_state) = yy_current_state;
1227:         (yy_last_accepting_cpos) = yy_cp;
1228:     }
1229: while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
1230:     {
1231:         yy_current_state = (int) yy_def[yy_current_state];
1232:         if ( yy_current_state >= 145 )
1233:             yy_c = yy_meta[(unsigned int) yy_c];
1234:     }
1235: yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
1236: yy_is_jam = (yy_current_state == 144);
1237:
1238: return yy_is_jam ? 0 : yy_current_state;
1239: }
1240:
1241: #ifndef YY_NO_INPUT
1242: #ifdef __cplusplus
1243:     static int yyinput (void)
1244: #else
1245:     static int input (void)
1246: #endif
1247:
1248: {
1249:     int c;
1250:
1251:     *(yy_c_buf_p) = (yy_hold_char);
1252:
1253:     if ( *(yy_c_buf_p) == YY_END_OF_BUFFER_CHAR )
1254:     {
1255:         /* yy_c_buf_p now points to the character we want to return.
1256:          * If this occurs *before* the EOB characters, then it's a
1257:          * valid NUL; if not, then we've hit the end of the buffer.
1258:          */
1259:         if ( (yy_c_buf_p) < &YY_CURRENT_BUFFER_LVALUE->yy_ch_buf[(yy_n_chars)] )
1260:             /* This was really a NUL. */
1261:             *(yy_c_buf_p) = '\0';
1262:

```

```

1263:     else
1264:         { /* need more input */
1265:             int offset = (yy_c_buf_p) - (yytext_ptr);
1266:             ++(yy_c_buf_p);
1267:
1268:             switch ( yy_get_next_buffer( ) )
1269:             {
1270:                 case EOB_ACT_LAST_MATCH:
1271:                     /* This happens because yy_g_n_b()
1272:                      * sees that we've accumulated a
1273:                      * token and flags that we need to
1274:                      * try matching the token before
1275:                      * proceeding.  But for input(),
1276:                      * there's no matching to consider.
1277:                      * So convert the EOB_ACT_LAST_MATCH
1278:                      * to EOB_ACT_END_OF_FILE.
1279:                      */
1280:
1281:                     /* Reset buffer status. */
1282:                     MessageManipulatorrestart(MessageManipulatorin );
1283:
1284:                     /*FALLTHROUGH*/
1285:
1286:                 case EOB_ACT_END_OF_FILE:
1287:                     {
1288:                         if ( MessageManipulatorwrap( ) )
1289:                             return EOF;
1290:
1291:                         if ( ! (yy_did_buffer_switch_on_eof) )
1292:                             YY_NEW_FILE;
1293: #ifdef __cplusplus
1294:                             return yyinput();
1295: #else
1296:                             return input();
1297: #endif
1298:                     }
1299:
1300:                 case EOB_ACT_CONTINUE_SCAN:
1301:                     (yy_c_buf_p) = (yytext_ptr) + offset;
1302:                     break;
1303:             }

```



```

1304:     }
1305: }
1306:
1307: c = *(unsigned char *) (yy_c_buf_p);    /* cast for 8-bit char's */
1308: *(yy_c_buf_p) = '\0';    /* preserve MessageManipulator text */
1309: (yy_hold_char) = *++(yy_c_buf_p);
1310:
1311: return c;
1312: }
1313: #endif    /* ifndef YY_NO_INPUT */
1314:
1315: /** Immediately switch to a different input stream.
1316:  * @param input_file A readable stream.
1317:  *
1318:  * @note This function does not reset the start condition to @c INITIAL .
1319:  */
1320: void MessageManipulatorrestart (FILE * input_file )
1321: {
1322:
1323: if ( ! YY_CURRENT_BUFFER ){
1324:     MessageManipulatorensure_buffer_stack ();
1325:     YY_CURRENT_BUFFER_LVALUE =
1326:         MessageManipulator_create_buffer(MessageManipulatorin,YY_BUF_SIZE );
1327: }
1328:
1329: MessageManipulator_init_buffer(YY_CURRENT_BUFFER,input_file );
1330: MessageManipulator_load_buffer_state( );
1331: }
1332:
1333: /** Switch to a different input buffer.
1334:  * @param new_buffer The new input buffer.
1335:  *
1336:  */
1337: void MessageManipulator_switch_to_buffer (YY_BUFFER_STATE new_buffer )
1338: {
1339:
1340: /* TODO. We should be able to replace this entire function body
1341:  * with
1342:  *     MessageManipulatorpop_buffer_state();
1343:  *     MessageManipulatorpush_buffer_state(new_buffer);
1344:  */

```

```

1345: MessageManipulatorensure_buffer_stack ();
1346: if ( YY_CURRENT_BUFFER == new_buffer )
1347:     return;
1348:
1349: if ( YY_CURRENT_BUFFER )
1350:     {
1351:         /* Flush out information for old buffer. */
1352:         *(yy_c_buf_p) = (yy_hold_char);
1353:         YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
1354:         YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
1355:     }
1356:
1357: YY_CURRENT_BUFFER_LVALUE = new_buffer;
1358: MessageManipulator_load_buffer_state( );
1359:
1360: /* We don't actually know whether we did this switch during
1361:  * EOF (MessageManipulatorwrap()) processing, but the only time this flag
1362:  * is looked at is after MessageManipulatorwrap() is called, so it's safe
1363:  * to go ahead and always set it.
1364:  */
1365: (yy_did_buffer_switch_on_eof) = 1;
1366: }
1367:
1368: static void MessageManipulator_load_buffer_state (void)
1369: {
1370:     (yy_n_chars) = YY_CURRENT_BUFFER_LVALUE->yy_n_chars;
1371:     (yytext_ptr) = (yy_c_buf_p) = YY_CURRENT_BUFFER_LVALUE->yy_buf_pos;
1372:     MessageManipulatorin = YY_CURRENT_BUFFER_LVALUE->yy_input_file;
1373:     (yy_hold_char) = *(yy_c_buf_p);
1374: }
1375:
1376: /** Allocate and initialize an input buffer state.
1377:  * @param file A readable stream.
1378:  * @param size The character buffer size in bytes. When in doubt, use @c YY_BUF_SIZE.
1379:  *
1380:  * @return the allocated buffer state.
1381:  */
1382: YY_BUFFER_STATE MessageManipulator_create_buffer (FILE * file, int size )
1383: {
1384:     YY_BUFFER_STATE b;
1385:

```

```

1386: b = (YY_BUFFER_STATE) MessageManipulatoralloc(sizeof( struct yy_buffer_state ) );
1387: if ( ! b )
1388:     YY_FATAL_ERROR( "out of dynamic memory in MessageManipulator_create_buffer()" );
1389:
1390: b->yy_buf_size = size;
1391:
1392: /* yy_ch_buf has to be 2 characters longer than the size given because
1393:  * we need to put in 2 end-of-buffer characters.
1394:  */
1395: b->yy_ch_buf = (char *) MessageManipulatoralloc(b->yy_buf_size + 2 );
1396: if ( ! b->yy_ch_buf )
1397:     YY_FATAL_ERROR( "out of dynamic memory in MessageManipulator_create_buffer()" );
1398:
1399: b->yy_is_our_buffer = 1;
1400:
1401: MessageManipulator_init_buffer(b,file );
1402:
1403: return b;
1404: }
1405:
1406: /** Destroy the buffer.
1407:  * @param b a buffer created with MessageManipulator_create_buffer()
1408:  *
1409:  */
1410: void MessageManipulator_delete_buffer (YY_BUFFER_STATE b )
1411: {
1412:
1413: if ( ! b )
1414:     return;
1415:
1416: if ( b == YY_CURRENT_BUFFER ) /* Not sure if we should pop here. */
1417:     YY_CURRENT_BUFFER_LVALUE = (YY_BUFFER_STATE) 0;
1418:
1419: if ( b->yy_is_our_buffer )
1420:     MessageManipulatorfree((void *) b->yy_ch_buf );
1421:
1422: MessageManipulatorfree((void *) b );
1423: }
1424:
1425: #ifndef __cplusplus
1426: extern int isatty (int );

```

```

1427: #endif /* __cplusplus */
1428:
1429: /* Initializes or reinitializes a buffer.
1430:  * This function is sometimes called more than once on the same buffer,
1431:  * such as during a MessageManipulatorrestart() or at EOF.
1432:  */
1433: static void MessageManipulator_init_buffer (YY_BUFFER_STATE b, FILE * file )
1434:
1435: {
1436:     int oerrno = errno;
1437:
1438:     MessageManipulator_flush_buffer(b );
1439:
1440:     b->yy_input_file = file;
1441:     b->yy_fill_buffer = 1;
1442:
1443:     /* If b is the current buffer, then MessageManipulator_init_buffer was _probably_
1444:      * called from MessageManipulatorrestart() or through yy_get_next_buffer.
1445:      * In that case, we don't want to reset the lineno or column.
1446:      */
1447:     if (b != YY_CURRENT_BUFFER){
1448:         b->yy_bs_lineno = 1;
1449:         b->yy_bs_column = 0;
1450:     }
1451:
1452:     b->yy_is_interactive = file ? (isatty( fileno(file) ) > 0) : 0;
1453:
1454:     errno = oerrno;
1455: }
1456:
1457: /** Discard all buffered characters. On the next scan, YY_INPUT will be called.
1458:  * @param b the buffer state to be flushed, usually @c YY_CURRENT_BUFFER.
1459:  *
1460:  */
1461: void MessageManipulator_flush_buffer (YY_BUFFER_STATE b )
1462: {
1463:     if ( ! b )
1464:         return;
1465:
1466:     b->yy_n_chars = 0;
1467:

```

```

1468:  /* We always need two end-of-buffer characters. The first causes
1469:   * a transition to the end-of-buffer state. The second causes
1470:   * a jam in that state.
1471:   */
1472:  b->yy_ch_buf[0] = YY_END_OF_BUFFER_CHAR;
1473:  b->yy_ch_buf[1] = YY_END_OF_BUFFER_CHAR;
1474:
1475:  b->yy_buf_pos = &b->yy_ch_buf[0];
1476:
1477:  b->yy_at_bol = 1;
1478:  b->yy_buffer_status = YY_BUFFER_NEW;
1479:
1480:  if ( b == YY_CURRENT_BUFFER )
1481:      MessageManipulator_load_buffer_state( );
1482: }
1483:
1484: /** Pushes the new state onto the stack. The new state becomes
1485:  * the current state. This function will allocate the stack
1486:  * if necessary.
1487:  * @param new_buffer The new state.
1488:  *
1489:  */
1490: void MessageManipulatorpush_buffer_state (YY_BUFFER_STATE new_buffer )
1491: {
1492:     if (new_buffer == NULL)
1493:         return;
1494:
1495:     MessageManipulatorensure_buffer_stack();
1496:
1497:     /* This block is copied from MessageManipulator_switch_to_buffer. */
1498:     if ( YY_CURRENT_BUFFER )
1499:     {
1500:         /* Flush out information for old buffer. */
1501:         *(yy_c_buf_p) = (yy_hold_char);
1502:         YY_CURRENT_BUFFER_LVALUE->yy_buf_pos = (yy_c_buf_p);
1503:         YY_CURRENT_BUFFER_LVALUE->yy_n_chars = (yy_n_chars);
1504:     }
1505:
1506:     /* Only push if top exists. Otherwise, replace top. */
1507:     if (YY_CURRENT_BUFFER)
1508:         (yy_buffer_stack_top)++;

```

```

1509: YY_CURRENT_BUFFER_LVALUE = new_buffer;
1510:
1511: /* copied from MessageManipulator_switch_to_buffer. */
1512: MessageManipulator_load_buffer_state( );
1513: (yy_did_buffer_switch_on_eof) = 1;
1514: }
1515:
1516: /** Removes and deletes the top of the stack, if present.
1517: * The next element becomes the new top.
1518: *
1519: */
1520: void MessageManipulatorpop_buffer_state (void)
1521: {
1522:     if (!YY_CURRENT_BUFFER)
1523:         return;
1524:
1525:     MessageManipulator_delete_buffer(YY_CURRENT_BUFFER );
1526:     YY_CURRENT_BUFFER_LVALUE = NULL;
1527:     if ((yy_buffer_stack_top) > 0)
1528:         --(yy_buffer_stack_top);
1529:
1530:     if (YY_CURRENT_BUFFER) {
1531:         MessageManipulator_load_buffer_state( );
1532:         (yy_did_buffer_switch_on_eof) = 1;
1533:     }
1534: }
1535:
1536: /* Allocates the stack if it does not exist.
1537: * Guarantees space for at least one push.
1538: */
1539: static void MessageManipulatorensure_buffer_stack (void)
1540: {
1541:     int num_to_alloc;
1542:
1543:     if (!(yy_buffer_stack)) {
1544:
1545:         /* First allocation is just for 2 elements, since we don't know if this
1546:          * scanner will even need a stack. We use 2 instead of 1 to avoid an
1547:          * immediate realloc on the next call.
1548:          */
1549:         num_to_alloc = 1;

```

```

1550:     (yy_buffer_stack) = (struct yy_buffer_state**)MessageManipulatoralloc
1551:         (num_to_alloc * sizeof(struct yy_buffer_state*))
1552:     );
1553:
1554:     memset((yy_buffer_stack), 0, num_to_alloc * sizeof(struct yy_buffer_state*));
1555:
1556:     (yy_buffer_stack_max) = num_to_alloc;
1557:     (yy_buffer_stack_top) = 0;
1558:     return;
1559: }
1560:
1561: if ((yy_buffer_stack_top) >= ((yy_buffer_stack_max)) - 1){
1562:
1563:     /* Increase the buffer to prepare for a possible push. */
1564:     int grow_size = 8 /* arbitrary grow size */;
1565:
1566:     num_to_alloc = (yy_buffer_stack_max) + grow_size;
1567:     (yy_buffer_stack) = (struct yy_buffer_state**)MessageManipulatorrealloc
1568:         ((yy_buffer_stack),
1569:         num_to_alloc * sizeof(struct yy_buffer_state*)
1570:         );
1571:
1572:     /* zero only the new slots.*/
1573:     memset((yy_buffer_stack) + (yy_buffer_stack_max), 0, grow_size * sizeof(struct
yy_buffer_state*));
1574:     (yy_buffer_stack_max) = num_to_alloc;
1575: }
1576: }
1577:
1578: /** Setup the input buffer state to scan directly from a user-specified character buffer.
1579: * @param base the character buffer
1580: * @param size the size in bytes of the character buffer
1581: *
1582: * @return the newly allocated buffer state object.
1583: */
1584: YY_BUFFER_STATE MessageManipulator_scan_buffer (char * base, yy_size_t size )
1585: {
1586:     YY_BUFFER_STATE b;
1587:
1588:     if ( size < 2 ||
1589:         base[size-2] != YY_END_OF_BUFFER_CHAR ||

```

```

1590:     base[size-1] != YY_END_OF_BUFFER_CHAR )
1591:     /* They forgot to leave room for the EOB's. */
1592:     return 0;
1593:
1594: b = (YY_BUFFER_STATE) MessageManipulatoralloc(sizeof( struct yy_buffer_state ) );
1595: if ( ! b )
1596:     YY_FATAL_ERROR( "out of dynamic memory in MessageManipulator_scan_buffer()" );
1597:
1598: b->yy_buf_size = size - 2; /* "- 2" to take care of EOB's */
1599: b->yy_buf_pos = b->yy_ch_buf = base;
1600: b->yy_is_our_buffer = 0;
1601: b->yy_input_file = 0;
1602: b->yy_n_chars = b->yy_buf_size;
1603: b->yy_is_interactive = 0;
1604: b->yy_at_bol = 1;
1605: b->yy_fill_buffer = 0;
1606: b->yy_buffer_status = YY_BUFFER_NEW;
1607:
1608: MessageManipulator_switch_to_buffer(b );
1609:
1610: return b;
1611: }
1612:
1613: /** Setup the input buffer state to scan a string. The next call to MessageManipulatorlex() will
1614: * scan from a @e copy of @a str.
1615: * @param str a NUL-terminated string to scan
1616: *
1617: * @return the newly allocated buffer state object.
1618: * @note If you want to scan bytes that may contain NUL values, then use
1619: * MessageManipulator_scan_bytes() instead.
1620: */
1621: YY_BUFFER_STATE MessageManipulator_scan_string (yyconst char * yy_str )
1622: {
1623:
1624: return MessageManipulator_scan_bytes(yy_str,strlen(yy_str) );
1625: }
1626:
1627: /** Setup the input buffer state to scan the given bytes. The next call to MessageManipulatorlex()
will
1628: * scan from a @e copy of @a bytes.
1629: * @param bytes the byte buffer to scan

```



```

1630: * @param len the number of bytes in the buffer pointed to by @a bytes.
1631: *
1632: * @return the newly allocated buffer state object.
1633: */
1634: YY_BUFFER_STATE MessageManipulator_scan_bytes (yyconst char * bytes, int len )
1635: {
1636:     YY_BUFFER_STATE b;
1637:     char *buf;
1638:     yy_size_t n;
1639:     int i;
1640:
1641:     /* Get memory for full buffer, including space for trailing EOB's. */
1642:     n = len + 2;
1643:     buf = (char *) MessageManipulatoralloc(n );
1644:     if ( ! buf )
1645:         YY_FATAL_ERROR( "out of dynamic memory in MessageManipulator_scan_bytes()" );
1646:
1647:     for ( i = 0; i < len; ++i )
1648:         buf[i] = bytes[i];
1649:
1650:     buf[len] = buf[len+1] = YY_END_OF_BUFFER_CHAR;
1651:
1652:     b = MessageManipulator_scan_buffer(buf,n );
1653:     if ( ! b )
1654:         YY_FATAL_ERROR( "bad buffer in MessageManipulator_scan_bytes()" );
1655:
1656:     /* It's okay to grow etc. this buffer, and we should throw it
1657:      * away when we're done.
1658:      */
1659:     b->yy_is_our_buffer = 1;
1660:
1661:     return b;
1662: }
1663:
1664: #ifndef YY_EXIT_FAILURE
1665: #define YY_EXIT_FAILURE 2
1666: #endif
1667:
1668: static void yy_fatal_error (yyconst char* msg )
1669: {
1670:     (void) fprintf( stderr, "%s \n", msg );

```

```

1671:  exit( YY_EXIT_FAILURE );
1672: }
1673:
1674: /* Redefine yyless() so it works in section 3 code. */
1675:
1676: #undef yyless
1677: #define yyless(n) \
1678:  do \
1679:  { \
1680:    /* Undo effects of setting up MessageManipulator text. */ \
1681:    int yyless_macro_arg = (n); \
1682:    YY_LESS_LINENO(yyval_macro_arg); \
1683:    MessageManipulator text[MessageManipulatorleng] = (yy_hold_char); \
1684:    (yy_c_buf_p) = MessageManipulator text + yyless_macro_arg; \
1685:    (yy_hold_char) = *(yy_c_buf_p); \
1686:    *(yy_c_buf_p) = '\0'; \
1687:    MessageManipulatorleng = yyless_macro_arg; \
1688:  } \
1689:  while ( 0 )
1690:
1691: /* Accessor methods (get/set functions) to struct members. */
1692:
1693: /** Get the current line number.
1694:  *
1695:  */
1696: int MessageManipulatorget_lineno (void)
1697: {
1698:
1699:  return MessageManipulatorlineno;
1700: }
1701:
1702: /** Get the input stream.
1703:  *
1704:  */
1705: FILE *MessageManipulatorget_in (void)
1706: {
1707:  return MessageManipulatorin;
1708: }
1709:
1710: /** Get the output stream.
1711:  *

```

```

1712: */
1713: FILE *MessageManipulatorget_out (void)
1714: {
1715:     return MessageManipulatorout;
1716: }
1717:
1718: /** Get the length of the current token.
1719: *
1720: */
1721: int MessageManipulatorget_leng (void)
1722: {
1723:     return MessageManipulatorleng;
1724: }
1725:
1726: /** Get the current token.
1727: *
1728: */
1729:
1730: char *MessageManipulatorget_text (void)
1731: {
1732:     return MessageManipulatortext;
1733: }
1734:
1735: /** Set the current line number.
1736: * @param line_number
1737: *
1738: */
1739: void MessageManipulatorset_lineno (int line_number )
1740: {
1741:
1742:     MessageManipulatorlineno = line_number;
1743: }
1744:
1745: /** Set the input stream. This does not discard the current
1746: * input buffer.
1747: * @param in_str A readable stream.
1748: *
1749: * @see MessageManipulator_switch_to_buffer
1750: */
1751: void MessageManipulatorset_in (FILE * in_str )
1752: {

```

```

1753:     MessageManipulatorin = in_str ;
1754: }
1755:
1756: void MessageManipulatorset_out (FILE * out_str )
1757: {
1758:     MessageManipulatorout = out_str ;
1759: }
1760:
1761: int MessageManipulatorget_debug (void)
1762: {
1763:     return MessageManipulator_flex_debug;
1764: }
1765:
1766: void MessageManipulatorset_debug (int bdebug )
1767: {
1768:     MessageManipulator_flex_debug = bdebug ;
1769: }
1770:
1771: /* MessageManipulatorlex_destroy is for both reentrant and non-reentrant scanners. */
1772: int MessageManipulatorlex_destroy (void)
1773: {
1774:
1775:     /* Pop the buffer stack, destroying each element. */
1776:     while(YY_CURRENT_BUFFER){
1777:         MessageManipulator_delete_buffer(YY_CURRENT_BUFFER );
1778:         YY_CURRENT_BUFFER_LVALUE = NULL;
1779:         MessageManipulatorpop_buffer_state();
1780:     }
1781:
1782:     /* Destroy the stack itself. */
1783:     MessageManipulatorfree((yy_buffer_stack) );
1784:     (yy_buffer_stack) = NULL;
1785:
1786:     return 0;
1787: }
1788:
1789: /*
1790:  * Internal utility routines.
1791:  */
1792:
1793: #ifndef yytext_ptr

```

```

1794: static void yy_flex_strncpy (char* s1, yyconst char * s2, int n )
1795: {
1796:     register int i;
1797:     for ( i = 0; i < n; ++i )
1798:         s1[i] = s2[i];
1799: }
1800: #endif
1801:
1802: #ifdef YY_NEED_STRLEN
1803: static int yy_flex_strlen (yyconst char * s )
1804: {
1805:     register int n;
1806:     for ( n = 0; s[n]; ++n )
1807:         ;
1808:
1809:     return n;
1810: }
1811: #endif
1812:
1813: void *MessageManipulatoralloc (yy_size_t size )
1814: {
1815:     return (void *) malloc( size );
1816: }
1817:
1818: void *MessageManipulatorrealloc (void * ptr, yy_size_t size )
1819: {
1820:     /* The cast to (char *) in the following accommodates both
1821:      * implementations that use char* generic pointers, and those
1822:      * that use void* generic pointers.  It works with the latter
1823:      * because both ANSI C and C++ allow castless assignment from
1824:      * any pointer type to void*, and deal with argument conversions
1825:      * as though doing an assignment.
1826:      */
1827:     return (void *) realloc( (char *) ptr, size );
1828: }
1829:
1830: void MessageManipulatorfree (void * ptr )
1831: {
1832:     free( (char *) ptr ); /* see MessageManipulatorrealloc() for (char *) cast */
1833: }
1834:

```

```

1835: #define YYTABLES_NAME "yytables"
1836:
1837: #undef YY_NEW_FILE
1838: #undef YY_FLUSH_BUFFER
1839: #undef yy_set_bol
1840: #undef yy_new_buffer
1841: #undef yy_set_interactive
1842: #undef yytext_ptr
1843: #undef YY_DO_BEFORE_ACTION
1844:
1845: #ifdef YY_DECL_IS_OURS
1846: #undef YY_DECL_IS_OURS
1847: #undef YY_DECL
1848: #endif
1849: #line 45 "msgdef.l"
1850:
1851:
1852:
1853: int MessageManipulatorwrap() {return 1;}
1854:
1855: void MessageManipulatorerror(const char *str)
1856: {
1857:   fprintf(stderr,"MessageManipulator error: %s on token %s \n",str,MessageManipulatortext);
1858: }
1859:

```

File: sdm/common/MessageManipulator/Makefile

```
1: include ../../Makefile.common
2: include ../../$(MAKEFILE_DEFS)
3:
4: .PHONY:    all clean distclean
5:
6: all:  MessageManipulator.o lex.MessageManipulator.o msgdef.tab.o message.o
7:
8: MessageManipulator.o: MessageManipulator.cpp MessageManipulator.h message.h
9:  $(CXX) $(CXXFLAGS) -fPIC -c $<
10:
11: msgdefmain.o:  msgdefmain.c
12:  $(CC) $(CFLAGS) -c $<
13:
14: lex.MessageManipulator.o: lex.MessageManipulator.c msgdef.tab.c
15:  $(CC) $(CFLAGS) -fPIC -c $<
16:
17: msgdef.tab.o:  msgdef.tab.c message.h
18:  $(CC) $(CFLAGS) -fPIC -c $<
19:
20: lex.MessageManipulator.c:  msgdef.l msgdef.tab.c
21:  $(LEX) $(LEXFLAGS) -PMessageManipulator $<
22:
23: msgdef.tab.c:  msgdef.y message.h
24:  $(YACC) $(YACCFLAGS) -p MessageManipulator $<
25:
26: message.o:  message.c
27:  $(CC) $(CFLAGS) -fPIC -c $<
28:
29: clean:
30:  rm -f *.o *~
31:
32: distclean: clean
33:  rm -f msgdef.output
```

File: sdm/common/MessageManipulator/msgdef.l

```
1: %{
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include <string.h>
5: #include "msgdef.tab.h"
6: %}
7:
8: %option nounput
9:
10: %%
11:
12: "<"          {return LT_SY;}
13: "="          {return EQUAL_SY;}
14: "/"          {return SLASH_SY;}
15: ">"          {return GT_SY;}
16: ":"          {return COLON_SY;}
17:
18: "DataMsg"     {return DATA_MSG_SY;}
19: "CommandMsg"  {return CMD_MSG_SY;}
20: "FaultMsg"    {return FAULT_MSG_SY;}
21:
22: [dD][aA][tT][aA]_rR[eE][pP][lL][yY]_mM[sS][gG]_nN[aA][mM][eE] {return REPLY_SY;}
23: [cC][oO][mM][mM][aA][nN][dD]_mM[sS][gG]_nN[aA][mM][eE]      {return REPLY_SY;}
24: "Request"     {return RQST_SY;}
25: "Notification" {return NOTI_SY;}
26: "Command"     {return CMD_SY;}
27:
28: "id"          {return ID_SY;}
29:
30: "name"        {return NAME_SY;}
31: "Variable"    {return VAR_SY;}
32: "length"      {return LENGTH_SY;}
33: "NS"          {return NOT_SPECIFIED_SY;}
34:
35: "UINT08"|"INT08"|"UINT16"|"INT16"|"UINT32"|"INT32"|"FLOAT32"|"FLOAT64" {
MessageManipulatorlval.str = strdup(yytext); return DATATYPE; }
36:
37: \"[^\"]*\"      {MessageManipulatorlval.str = strdup(yytext+1, strlen(yytext)-2); return
STRING;}
```



```

38: [0-9]*[.][0-9]+      {MessageManipulatorlval.real = atof(yytext); return FLOAT;}
39: [0-9]+               {MessageManipulatorlval.integer = atoi(yytext); return INT;}
40:
41:
42: [ ]*|[ \t]*|[ \n]*    { /*ignore whitespace*/}
43: .                     {printf ("Invalid token \n");}
44:
45: %%
46:
47: int yywrap() {return 1;}
48:
49: void MessageManipulatorerror(const char *str)
50: {
51: fprintf(stderr,"MessageManipulator error: %s on token %s \n",str,yytext);
52: }

```

File: sdm/common/MessageManipulator/MessageManipulator.h

```
1: #ifndef __MESSAGE_MANIPULATOR_H_
2: #define __MESSAGE_MANIPULATOR_H_
3: #include <pthread.h>
4: #include "../message/SDMData.h"
5: #include "../message/SDMRegInfo.h"
6: #include "../message/SDMService.h"
7: #include "../message/SDMCommand.h"
8: #include "../message/SDMSerreqst.h"
9: #include "../message/SDMMessage_ID.h"
10:
11: extern "C"
12: {
13: #include "message.h"
14: }
15:
16: /*The SDMMessageManipulator class is intended to marshall and unmarshal arbitrary SDM messages
based on a message def*/
17:
18: class SDMLIB_API MessageManipulator
19: {
20: public:
21: MessageManipulator();
22: ~MessageManipulator();
23:
24: enum wrapper_type setMsgDef(char* msgdef);
25: enum wrapper_type setMsgDef(SDMRegInfo&);
26:
27: int getLength(enum msg_type type);
28:
29: bool isMsgValid(const SDMData& DatMsg, enum msg_type type);
30: unsigned char getUINT08Value(char* name, SDMData&, enum msg_type type, bool* IsValid =
NULL);
31: signed char getINT08Value(char* name, SDMData&, enum msg_type type, bool* IsValid = NULL);
32: unsigned short getUINT16Value(char* name, SDMData&, enum msg_type type, bool* IsValid =
NULL);
33: short getINT16Value(char* name, SDMData&, enum msg_type type, bool* IsValid = NULL);
34: unsigned long getUINT32Value(char* name, SDMData&, enum msg_type type, bool* IsValid =
NULL);
35: long getINT32Value(char* name, SDMData&, enum msg_type type, bool* IsValid = NULL);
36: float getFLOAT32Value(char* name, SDMData&, enum msg_type type, bool* IsValid = NULL);
```

```

37: double getFLOAT64Value(char* name, SDMDData&, enum msg_type type, bool* IsValid = NULL);
38:
39: bool isMsgValid(const SDMCommand& CmdMsg);
40: unsigned char getUINT08Value(char* name, SDMCommand&, bool* IsValid = NULL);
41: signed char getINT08Value(char* name, SDMCommand&, bool* IsValid = NULL);
42: unsigned short getUINT16Value(char* name, SDMCommand&, bool* IsValid = NULL);
43: short getINT16Value(char* name, SDMCommand&, bool* IsValid = NULL);
44: unsigned long getUINT32Value(char* name, SDMCommand&, bool* IsValid = NULL);
45: long getINT32Value(char* name, SDMCommand&, bool* IsValid = NULL);
46: float getFLOAT32Value(char* name, SDMCommand&, bool* IsValid = NULL);
47: double getFLOAT64Value(char* name, SDMCommand&, bool* IsValid = NULL);
48:
49: bool isMsgValid(const SDMSerreqst& ReqMsg);
50: unsigned char getUINT08Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
51: signed char getINT08Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
52: unsigned short getUINT16Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
53: short getINT16Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
54: unsigned long getUINT32Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
55: long getINT32Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
56: float getFLOAT32Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
57: double getFLOAT64Value(char* name, SDMSerreqst&, bool* IsValid = NULL);
58:
59: void* getArray(char* name, SDMDData &, enum msg_type type, int &length);
60: void* getArray(char* name, SDMService &, int &length);
61: void* getArray(char* name, SDMCommand &, int &length);
62:
63: bool setArray(char* name, SDMDData &, enum msg_type type, const void* buffer, int length);
64: bool setArray(char* name, SDMService &, const void* buffer, int length);
65: bool setArray(char* name, SDMCommand &, const void* buffer, int length);
66:
67: bool setValue(char* name, SDMService&, unsigned char value);
68: bool setValue(char* name, SDMService&, char value);
69: bool setValue(char* name, SDMService&, unsigned short value);
70: bool setValue(char* name, SDMService&, short value);
71: bool setValue(char* name, SDMService&, unsigned long value);
72: bool setValue(char* name, SDMService&, long value);
73: bool setValue(char* name, SDMService&, float value);
74: bool setValue(char* name, SDMService&, double value);
75:
76: bool setValue(char* name, SDMCommand&, unsigned char value);
77: bool setValue(char* name, SDMCommand&, char value);

```

```

78: bool setValue(char* name, SDMCommand&, unsigned short value);
79: bool setValue(char* name, SDMCommand&, short value);
80: bool setValue(char* name, SDMCommand&, unsigned long value);
81: bool setValue(char* name, SDMCommand&, long value);
82: bool setValue(char* name, SDMCommand&, float value);
83: bool setValue(char* name, SDMCommand&, double value);
84:
85: bool setValue(char* name, SDMDData&, unsigned char value, enum msg_type type);
86: bool setValue(char* name, SDMDData&, char value, enum msg_type type);
87: bool setValue(char* name, SDMDData&, unsigned short value, enum msg_type type);
88: bool setValue(char* name, SDMDData&, short value, enum msg_type type);
89: bool setValue(char* name, SDMDData&, unsigned long value, enum msg_type type);
90: bool setValue(char* name, SDMDData&, long value ,enum msg_type type);
91: bool setValue(char* name, SDMDData&, float value, enum msg_type type);
92: bool setValue(char* name, SDMDData&, double value, enum msg_type type);
93:
94: SDMMMessage_ID getMsgID(enum msg_type type);
95: enum wrapper_type getType() { return Msg.type; }
96:
97: void applyGetScaleFactor(bool apply);
98: private:
99: value_t getValue(char* name,SDMDData&,enum msg_type type, bool* IsValid);
100:  value_t getValue(char* name,SDMCommand&, bool* IsValid);
101:  value_t getValue(char* name,SDMSerreqst&, bool* IsValid);
102:  value_t getValue(const char* Buffer, var* Variable, bool* IsValid);
103:  bool setValue(char* name,SDMService& request,value_t val);
104:  bool setValue(char* name,SDMCommand& command,value_t val);
105:  bool setValue(char* name,SDMDData& data,value_t val,enum msg_type type);
106:  bool setValue(char* Buffer, var* Variable, value_t Value);
107:  bool isMsgValid(const char* Buffer, var* Variable);
108:  msg Msg;
109:  bool scaled;
110:  static pthread_mutex_t ParseMutex;
111: };
112:
113: #endif

```

File: sdm/common/MessageManipulator/msgdef.tab.h

```
1: /* A Bison parser, made by GNU Bison 1.875d. */
2:
3: /* Skeleton parser for Yacc-like parsing with Bison,
4:  Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.
5:
6:  This program is free software; you can redistribute it and/or modify
7:  it under the terms of the GNU General Public License as published by
8:  the Free Software Foundation; either version 2, or (at your option)
9:  any later version.
10:
11:  This program is distributed in the hope that it will be useful,
12:  but WITHOUT ANY WARRANTY; without even the implied warranty of
13:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14:  GNU General Public License for more details.
15:
16:  You should have received a copy of the GNU General Public License
17:  along with this program; if not, write to the Free Software
18:  Foundation, Inc., 59 Temple Place - Suite 330,
19:  Boston, MA 02111-1307, USA.  */
20:
21: /* As a special exception, when this file is copied by Bison into a
22:  Bison output file, you may use that output file without restriction.
23:  This special exception was added by the Free Software Foundation
24:  in version 1.24 of Bison.  */
25:
26: /* Tokens.  */
27: #ifndef YYTOKENTYPE
28: # define YYTOKENTYPE
29:  /* Put the tokens into the symbol table, so that GDB and other debuggers
30:   know about them.  */
31:  enum yytokentype {
32:    LT_SY = 258,
33:    CMD_MSG_SY = 259,
34:    DATA_MSG_SY = 260,
35:    NAME_SY = 261,
36:    EQUAL_SY = 262,
37:    SLASH_SY = 263,
38:    GT_SY = 264,
39:    COLON_SY = 265,
```

```

40:  VAR_SY = 266,
41:  LENGTH_SY = 267,
42:  COUNTMAX_SY = 268,
43:  RATEMAX_SY = 269,
44:  RQST_SY = 270,
45:  REPLY_SY = 271,
46:  ID_SY = 272,
47:  FAULT_MSG_SY = 273,
48:  CMD_SY = 274,
49:  NOTI_SY = 275,
50:  DATATYPE = 276,
51:  FLOAT = 277,
52:  INT = 278,
53:  STRING = 279,
54:  NOT_SPECIFIED_SY = 280
55:  };
56: #endif
57: #define LT_SY 258
58: #define CMD_MSG_SY 259
59: #define DATA_MSG_SY 260
60: #define NAME_SY 261
61: #define EQUAL_SY 262
62: #define SLASH_SY 263
63: #define GT_SY 264
64: #define COLON_SY 265
65: #define VAR_SY 266
66: #define LENGTH_SY 267
67: #define COUNTMAX_SY 268
68: #define RATEMAX_SY 269
69: #define RQST_SY 270
70: #define REPLY_SY 271
71: #define ID_SY 272
72: #define FAULT_MSG_SY 273
73: #define CMD_SY 274
74: #define NOTI_SY 275
75: #define DATATYPE 276
76: #define FLOAT 277
77: #define INT 278
78: #define STRING 279
79: #define NOT_SPECIFIED_SY 280
80:

```

```
81:
82:
83:
84: #if ! defined (YYSTYPE) && ! defined (YYSTYPE_IS_DECLARED)
85: #line 21 "msgdef.y"
86: typedef union YYSTYPE {
87: int integer;
88: float real;
89: char* str;
90: struct variable* var;
91: struct node_data* node;
92: } YYSTYPE;
93: /* Line 1285 of yacc.c. */
94: #line 95 "msgdef.tab.h"
95: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
96: # define YYSTYPE_IS_DECLARED 1
97: # define YYSTYPE_IS_TRIVIAL 1
98: #endif
99:
100: extern YYSTYPE MessageManipulatorlval;
101:
102:
103:
```

File: sdm/common/MessageManipulator/MessageManipulator.cpp

```
1: #include "MessageManipulator.h"
2: #include "../marshall.h"
3:
4: #include <string.h>
5:
6: pthread_mutex_t MessageManipulator::ParseMutex = PTHREAD_MUTEX_INITIALIZER;
7:
8: static value_t cast(value_t val, enum value_type val_type)
9: {
10: if(val.type != val_type) //then cast
11: {
12:     switch(val_type)
13:     {
14:     case _UINT08:
15:         switch(val.type)
16:         {
17:         case _UINT08:
18:             val.value.uint08 = (unsigned char)val.value.uint08;
19:             break;
20:         case _INT08:
21:             val.value.uint08 = (unsigned char)val.value.int08;
22:             break;
23:         case _UINT16:
24:             val.value.uint08 = (unsigned char)val.value.uint16;
25:             break;
26:         case _INT16:
27:             val.value.uint08 = (unsigned char)val.value.int16;
28:             break;
29:         case _UINT32:
30:             val.value.uint08 = (unsigned char)val.value.uint32;
31:             break;
32:         case _INT32:
33:             val.value.uint08 = (unsigned char)val.value.int32;
34:             break;
35:         case _FLOAT32:
36:             val.value.uint08 = (unsigned char)val.value.float32;
37:             break;
38:         case _FLOAT64:
39:             val.value.uint08 = (unsigned char)val.value.float64;
```



```

40:         break;
41:     case _EMPTY:
42:         break;
43:     }
44:     break;
45: case _INT08:
46:     switch(val.type)
47:     {
48:     case _UINT08:
49:         val.value.int08 = (char)val.value.uint08;
50:         break;
51:     case _INT08:
52:         val.value.int08 = (char)val.value.int08;
53:         break;
54:     case _UINT16:
55:         val.value.int08 = (char)val.value.uint16;
56:         break;
57:     case _INT16:
58:         val.value.int08 = (char)val.value.int16;
59:         break;
60:     case _UINT32:
61:         val.value.int08 = (char)val.value.uint32;
62:         break;
63:     case _INT32:
64:         val.value.int08 = (char)val.value.int32;
65:         break;
66:     case _FLOAT32:
67:         val.value.int08 = (char)val.value.float32;
68:         break;
69:     case _FLOAT64:
70:         val.value.int08 = (char)val.value.float64;
71:         break;
72:     case _EMPTY:
73:         break;
74:     }
75:     break;
76: case _UINT16:
77:     switch(val.type)
78:     {
79:     case _UINT08:
80:         val.value.uint16 = (unsigned short)val.value.uint08;

```

```

81:         break;
82:     case _INT08:
83:         val.value.uint16 = (unsigned short)val.value.int08;
84:         break;
85:     case _UINT16:
86:         val.value.uint16 = (unsigned short)val.value.uint16;
87:         break;
88:     case _INT16:
89:         val.value.uint16 = (unsigned short)val.value.int16;
90:         break;
91:     case _UINT32:
92:         val.value.uint16 = (unsigned short)val.value.uint32;
93:         break;
94:     case _INT32:
95:         val.value.uint16 = (unsigned short)val.value.int32;
96:         break;
97:     case _FLOAT32:
98:         val.value.uint16 = (unsigned short)val.value.float32;
99:         break;
100:    case _FLOAT64:
101:        val.value.uint16 = (unsigned short)val.value.float64;
102:        break;
103:    case _EMPTY:
104:        break;
105:    }
106:    break;
107: case _INT16:
108:     switch(val.type)
109:     {
110:     case _UINT08:
111:         val.value.int16 = (short)val.value.uint08;
112:         break;
113:     case _INT08:
114:         val.value.int16 = (short)val.value.int08;
115:         break;
116:     case _UINT16:
117:         val.value.int16 = (short)val.value.uint16;
118:         break;
119:     case _INT16:
120:         val.value.int16 = (short)val.value.int16;
121:         break;

```

```

122:         case _UINT32:
123:             val.value.int16 = (short)val.value.uint32;
124:             break;
125:         case _INT32:
126:             val.value.int16 = (short)val.value.int32;
127:             break;
128:         case _FLOAT32:
129:             val.value.int16 = (short)val.value.float32;
130:             break;
131:         case _FLOAT64:
132:             val.value.int16 = (short)val.value.float64;
133:             break;
134:         case _EMPTY:
135:             break;
136:     }
137:     break;
138: case _UINT32:
139:     switch(val.type)
140:     {
141:         case _UINT08:
142:             val.value.uint32 = (unsigned long)val.value.uint08;
143:             break;
144:         case _INT08:
145:             val.value.uint32 = (unsigned long)val.value.int08;
146:             break;
147:         case _UINT16:
148:             val.value.uint32 = (unsigned long)val.value.uint16;
149:             break;
150:         case _INT16:
151:             val.value.uint32 = (unsigned long)val.value.int16;
152:             break;
153:         case _UINT32:
154:             val.value.uint32 = (unsigned long)val.value.uint32;
155:             break;
156:         case _INT32:
157:             val.value.uint32 = (unsigned long)val.value.int32;
158:             break;
159:         case _FLOAT32:
160:             val.value.uint32 = (unsigned long)val.value.float32;
161:             break;
162:         case _FLOAT64:

```

```

163:         val.value.uint32 = (unsigned long)val.value.float64;
164:         break;
165:     case _EMPTY:
166:         break;
167:     }
168:     break;
169: case _INT32:
170:     switch(val.type)
171:     {
172:     case _UINT08:
173:         val.value.int32 = (long)val.value.uint08;
174:         break;
175:     case _INT08:
176:         val.value.int32 = (long)val.value.int08;
177:         break;
178:     case _UINT16:
179:         val.value.int32 = (long)val.value.uint16;
180:         break;
181:     case _INT16:
182:         val.value.int32 = (long)val.value.int16;
183:         break;
184:     case _UINT32:
185:         val.value.int32 = (long)val.value.uint32;
186:         break;
187:     case _INT32:
188:         val.value.int32 = (long)val.value.int32;
189:         break;
190:     case _FLOAT32:
191:         val.value.int32 = (long)val.value.float32;
192:         break;
193:     case _FLOAT64:
194:         val.value.int32 = (long)val.value.float64;
195:         break;
196:     case _EMPTY:
197:         break;
198:     }
199:     break;
200: case _FLOAT32:
201:     switch(val.type)
202:     {
203:     case _UINT08:

```

```

204:         val.value.float32 = (float)val.value.uint08;
205:         break;
206:     case _INT08:
207:         val.value.float32 = (float)val.value.int08;
208:         break;
209:     case _UINT16:
210:         val.value.float32 = (float)val.value.uint16;
211:         break;
212:     case _INT16:
213:         val.value.float32 = (float)val.value.int16;
214:         break;
215:     case _UINT32:
216:         val.value.float32 = (float)val.value.uint32;
217:         break;
218:     case _INT32:
219:         val.value.float32 = (float)val.value.int32;
220:         break;
221:     case _FLOAT32:
222:         val.value.float32 = (float)val.value.float32;
223:         break;
224:     case _FLOAT64:
225:         val.value.float32 = (float)val.value.float64;
226:         break;
227:     case _EMPTY:
228:         break;
229:     }
230:     break;
231: case _FLOAT64:
232:     switch(val.type)
233:     {
234:     case _UINT08:
235:         val.value.float64 = (double)val.value.uint08;
236:         break;
237:     case _INT08:
238:         val.value.float64 = (double)val.value.int08;
239:         break;
240:     case _UINT16:
241:         val.value.float64 = (double)val.value.uint16;
242:         break;
243:     case _INT16:
244:         val.value.float64 = (double)val.value.int16;

```

```

245:         break;
246:     case _UINT32:
247:         val.value.float64 = (double)val.value.uint32;
248:         break;
249:     case _INT32:
250:         val.value.float64 = (double)val.value.int32;
251:         break;
252:     case _FLOAT32:
253:         val.value.float64 = (double)val.value.float32;
254:         break;
255:     case _FLOAT64:
256:         val.value.float64 = (double)val.value.float64;
257:         break;
258:     case _EMPTY:
259:         break;
260:     }
261:     break;
262: case _EMPTY:
263:     break;
264: }
265: }
266: return val;
267: }
268:
269: MessageManipulator::MessageManipulator():Msg(),scaled(false)
270: {
271:     memset(&Msg,0,sizeof(msg));
272: }
273:
274: MessageManipulator::~~MessageManipulator()
275: {
276:     //Free the message defs tree
277:     deleteMessage(&Msg);
278: }
279: enum wrapper_type MessageManipulator::setMsgDef(char* msgdef)
280: {
281:     pthread_mutex_lock(&ParseMutex);
282:     parse(msgdef,&Msg);
283:     pthread_mutex_unlock(&ParseMutex);
284:     return Msg.type;
285: }

```

```

286:
287: enum wrapper_type MessageManipulator::setMsgDef(SDMRegInfo& info)
288: {
289:     return setMsgDef(info.msg_def);
290: }
291:
292: //////////////////////////////////////
293: //
294: //      GET FUNCTIONS
295: //
296: //////////////////////////////////////
297:
298: int MessageManipulator::getLength(enum msg_type type)
299: {
300:     switch(type)
301:     {
302:         case COMMANDMSG:
303:             if (Msg.command == NULL) return -1;
304:             return Msg.command->length;
305:             break;
306:         case DATAMSG:
307:             if (Msg.data == NULL) return -1;
308:             return Msg.data->length;
309:             break;
310:         case FAULTMSG:
311:             if (Msg.fault == NULL) return -1;
312:             return Msg.fault->length;
313:             break;
314:     }
315:     return -1;
316: }
317:
318: void* MessageManipulator::getArray(char* name,SDMData& dat,enum msg_type type,int& length)
319: {
320:     var* temp = NULL;
321:
322:     if (type == DATAMSG)
323:     {
324:         if (Msg.data == NULL)
325:             return NULL;
326:         else

```

```

327:         temp = find(name,Msg.data->list);
328:     }
329:     else if (type == FAULTMSG)
330:     {
331:         if (Msg.fault == NULL)
332:             return NULL;
333:         else
334:             temp = find(name,Msg.fault->list);
335:     }
336:
337:     if(temp != NULL)
338:     {
339:         length = temp->length;
340:         return (void*)(dat.msg + temp->start);
341:     }
342:     return NULL;
343: }
344:
345: void* MessageManipulator::getArray(char* name,SDMService& ser,int& length)
346: {
347:     var* temp;
348:
349:     if (Msg.command == NULL) return NULL;
350:     temp = find(name,Msg.command->list);
351:     if (temp != NULL)
352:     {
353:         length = temp->length;
354:         return (void*)(ser.data + temp->start);
355:     }
356:     return NULL;
357: }
358:
359: void* MessageManipulator::getArray(char* name,SDMCommand& cmd,int& length)
360: {
361:     return getArray(name,static_cast<SDMService&>(cmd),length);
362: }
363:
364: value_t MessageManipulator::getValue(const char* Buffer, var* Variable, bool* IsValid)
365: {
366:     value_t val;
367:     val.type = _EMPTY;

```



```

368: val.value.uint32 = 0;
369:
370:     if (Buffer == NULL || Variable == NULL)
371:     {
372:         return val;
373:     }
374:
375:     int start = Variable->start;
376:
377:     if (IsValid != NULL)
378:     {
379:         *IsValid = true;
380:     }
381:
382:     switch(Variable->type)
383:     {
384:         case _UINT08:
385:             val.type = _UINT08;
386:             val.value.uint08 = GET_UCHAR(&(Buffer[start]));
387:
388:             // If the value is equal to its invalid value, and an invalid value has been specified
389:             if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.uint08
== Variable->invalid_data.value.uint08)
390:                 *IsValid = false;
391:
392:             // Check if the value should be scaled
393:             if(scaled)
394:                 val.value.uint08 = static_cast<unsigned char>(val.value.uint08 * Variable-
>scale_factor);
395:             break;
396:         case _INT08:
397:             val.type = _INT08;
398:             val.value.int08 = GET_CHAR(&(Buffer[start]));
399:
400:             // If the value is equal to its invalid value, and an invalid value has been specified
401:             if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.int08
== Variable->invalid_data.value.int08)
402:                 *IsValid = false;
403:
404:             // Check if the value should be scaled
405:             if(scaled)
406:                 val.value.int08 = static_cast<char>(val.value.int08 * Variable->scale_factor);

```

```

407:         break;
408:     case _UINT16:
409:         val.type = _UINT16;
410:         val.value.uint16 = GET_USHORT(&(Buffer[start]));
411:
412:         // If the value is equal to its invalid value, and an invalid value has been specified
413:         if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.uint16
== Variable->invalid_data.value.uint16)
414:             *IsValid = false;
415:
416:         // Check if the value should be scaled
417:         if(scaled)
418:             val.value.uint16 = static_cast<unsigned short>(val.value.uint16 * Variable-
>scale_factor);
419:         break;
420:     case _INT16:
421:         val.type = _INT16;
422:         val.value.int16 = GET_SHORT(&(Buffer[start]));
423:
424:         // If the value is equal to its invalid value, and an invalid value has been specified
425:         if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.int08
== Variable->invalid_data.value.int08)
426:             *IsValid = false;
427:
428:         // Check if the value should be scaled
429:         if(scaled)
430:             val.value.int16 = static_cast<short>(val.value.int16 * Variable->scale_factor);
431:         break;
432:     case _UINT32:
433:         val.type = _UINT32;
434:         val.value.uint32 = GET_ULONG(&(Buffer[start]));
435:
436:         // If the value is equal to its invalid value, and an invalid value has been specified
437:         if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.uint32
== Variable->invalid_data.value.uint32)
438:             *IsValid = false;
439:
440:         // Check if the value should be scaled
441:         if(scaled)
442:             val.value.uint32 = static_cast<unsigned long>(val.value.uint32 * Variable-
>scale_factor);
443:         break;

```

```

444:     case _INT32:
445:         val.type = _INT32;
446:         val.value.int32 = GET_LONG(&(Buffer[start]));
447:
448:         // If the value is equal to its invalid value, and an invalid value has been specified
449:         if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.int32
== Variable->invalid_data.value.int32)
450:             *IsValid = false;
451:
452:         // Check if the value should be scaled
453:         if(scaled)
454:             val.value.int32 = static_cast<long>(val.value.int32 * Variable->scale_factor);
455:         break;
456:     case _FLOAT32:
457:         val.type = _FLOAT32;
458:         val.value.float32 = GET_FLOAT(&(Buffer[start]));
459:
460:         // If the value is equal to its invalid value, and an invalid value has been specified
461:         if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.float32
== Variable->invalid_data.value.float32)
462:             *IsValid = false;
463:
464:         // Check if the value should be scaled
465:         if(scaled)
466:             val.value.float32 = static_cast<float>(val.value.float32 * Variable->scale_factor);
467:         break;
468:     case _FLOAT64:
469:         val.type = _FLOAT64;
470:         val.value.float64 = GET_DOUBLE(&(Buffer[start]));
471:
472:         // If the value is equal to its invalid value, and an invalid value has been specified
473:         if (IsValid != NULL && Variable->invalid_data.type != _EMPTY && val.value.float64
== Variable->invalid_data.value.float64)
474:             *IsValid = false;
475:
476:         // Check if the value should be scaled
477:         if(scaled)
478:             val.value.float64 = static_cast<double>(val.value.float64 * Variable->scale_factor);
479:         break;
480:     case _EMPTY:
481:         break;
482: }

```

```

483:    return val;
484: }
485:
486: //////////////// Gets for SDMDData ////////////////
487:
488: bool MessageManipulator::isMsgValid(const char* MsgBuffer, var* Variable)
489: {
490:     if (Variable == NULL || MsgBuffer == NULL)
491:         return false;
492:     bool ThisValid=true, LeftValid=true, RightValid=true;
493:
494:     if (Variable->left != NULL)
495:         LeftValid = isMsgValid(MsgBuffer, Variable->left);
496:
497:     getValue(MsgBuffer, Variable, &ThisValid);
498:
499:     if (Variable->right != NULL)
500:         RightValid = isMsgValid(MsgBuffer, Variable->right);
501:
502:     return (ThisValid && LeftValid && RightValid);
503: }
504:
505: bool MessageManipulator::isMsgValid(const SDMDData& DatMsg, enum msg_type type)
506: {
507:     bool Result;
508:     if (type == DATAMSG)
509:         Result = isMsgValid(DatMsg.msg, Msg.data->list);
510:     else
511:         Result = isMsgValid(DatMsg.msg, Msg.fault->list);
512:     return Result;
513: }
514:
515: value_t MessageManipulator::getValue(char* name, SDMDData& dat, enum msg_type type, bool*
IsValid)
516: {
517:     var* temp;
518:
519:     if (type == DATAMSG)
520:         temp = find(name, Msg.data->list);
521:     else
522:         temp = find(name, Msg.fault->list);

```

```

523:    return getValue(dat.msg, temp, IsValid);
524: }
525:
526: unsigned char MessageManipulator::getUINT08Value(char* name,SDMData& dat, enum msg_type
type, bool* IsValid)
527: {
528:     value_t temp;
529:     temp = getValue(name,dat,type,IsValid);
530:     temp = cast(temp,_UINT08);
531:     return temp.value.uint08;
532: }
533:
534: signed char MessageManipulator::getINT08Value(char* name,SDMData& dat, enum msg_type
type, bool* IsValid)
535: {
536:     value_t temp;
537:     temp = getValue(name,dat,type,IsValid);
538:     temp = cast(temp,_INT08);
539:     return temp.value.int08;
540: }
541:
542: unsigned short MessageManipulator::getUINT16Value(char* name,SDMData& dat, enum msg_type
type, bool* IsValid)
543: {
544:     value_t temp;
545:     temp = getValue(name,dat,type,IsValid);
546:     temp = cast(temp,_UINT16);
547:     return temp.value.uint16;
548: }
549:
550: short MessageManipulator::getINT16Value(char* name,SDMData& dat, enum msg_type type,
bool* IsValid)
551: {
552:     value_t temp;
553:     temp = getValue(name,dat,type,IsValid);
554:     temp = cast(temp,_INT16);
555:     return temp.value.int16;
556: }
557:
558: unsigned long MessageManipulator::getUINT32Value(char* name,SDMData& dat, enum msg_type
type, bool* IsValid)
559: {

```

```

560:     value_t temp;
561:     temp = getValue(name,dat,type,IsValid);
562:     temp = cast(temp,_UINT32);
563:     return temp.value.uint32;
564: }
565:
566: long MessageManipulator::getINT32Value(char* name,SDMData& dat, enum msg_type type, bool*
IsValid)
567: {
568:     value_t temp;
569:     temp = getValue(name,dat,type,IsValid);
570:     temp = cast(temp,_INT32);
571:     return temp.value.int32;
572: }
573:
574: float MessageManipulator::getFLOAT32Value(char* name,SDMData& dat, enum msg_type type,
bool* IsValid)
575: {
576:     value_t temp;
577:     temp = getValue(name,dat,type,IsValid);
578:     temp = cast(temp,_FLOAT32);
579:     return temp.value.float32;
580: }
581:
582: double MessageManipulator::getFLOAT64Value(char* name,SDMData& dat, enum msg_type type,
bool* IsValid)
583: {
584:     value_t temp;
585:     temp = getValue(name,dat,type,IsValid);
586:     temp = cast(temp,_FLOAT64);
587:     return temp.value.float64;
588: }
589:
590: //////////// Gets for SDMCommand ////////////
591:
592: bool MessageManipulator::isMsgValid(const SDMCommand& CmdMsg)
593: {
594:     return isMsgValid (CmdMsg.data, Msg.command->list);
595: }
596:
597: value_t MessageManipulator::getValue(char* name, SDMCommand& com, bool* IsValid)
598: {

```

```

599:   var* temp;
600:
601:   temp = find(name,Msg.command->list);
602:   return getValue(com.data, temp, IsValid);
603: }
604:
605: unsigned char MessageManipulator::getUINT08Value(char* name,SDMCommand& com, bool*
IsValid)
606: {
607:   value_t temp;
608:   temp = getValue(name,com,IsValid);
609:   temp = cast(temp,_UINT08);
610:   return temp.value.uint08;
611: }
612:
613: signed char MessageManipulator::getINT08Value(char* name,SDMCommand& com, bool*
IsValid)
614: {
615:   value_t temp;
616:   temp = getValue(name,com,IsValid);
617:   temp = cast(temp,_INT08);
618:   return temp.value.int08;
619: }
620:
621: unsigned short MessageManipulator::getUINT16Value(char* name,SDMCommand& com, bool*
IsValid)
622: {
623:   value_t temp;
624:   temp = getValue(name,com,IsValid);
625:   temp = cast(temp,_UINT16);
626:   return temp.value.uint16;
627: }
628:
629: short MessageManipulator::getINT16Value(char* name,SDMCommand& com, bool* IsValid)
630: {
631:   value_t temp;
632:   temp = getValue(name,com,IsValid);
633:   temp = cast(temp,_INT16);
634:   return temp.value.int16;
635: }
636:

```

```

637: unsigned long MessageManipulator::getUINT32Value(char* name,SDMCommand& com, bool*
IsValid)
638: {
639:     value_t temp;
640:     temp = getValue(name,com,IsValid);
641:     temp = cast(temp,_UINT32);
642:     return temp.value.uint32;
643: }
644:
645: long MessageManipulator::getINT32Value(char* name,SDMCommand& com, bool* IsValid)
646: {
647:     value_t temp;
648:     temp = getValue(name,com,IsValid);
649:     temp = cast(temp,_INT32);
650:     return temp.value.int32;
651: }
652:
653: float MessageManipulator::getFLOAT32Value(char* name,SDMCommand& com, bool* IsValid)
654: {
655:     value_t temp;
656:     temp = getValue(name,com,IsValid);
657:     temp = cast(temp,_FLOAT32);
658:     return temp.value.float32;
659: }
660:
661: double MessageManipulator::getFLOAT64Value(char* name,SDMCommand& com, bool* IsValid)
662: {
663:     value_t temp;
664:     temp = getValue(name,com,IsValid);
665:     temp = cast(temp,_FLOAT64);
666:     return temp.value.float64;
667: }
668:
669: //////////////// Gets for SDMSerreqst ////////////////
670:
671: bool MessageManipulator::isMsgValid(const SDMSerreqst& ReqMsg)
672: {
673:     return isMsgValid(ReqMsg.data, Msg.command->list);
674: }
675:
676: value_t MessageManipulator::getValue(char* name, SDMSerreqst& req, bool* IsValid)

```



```

677: {
678:     var* temp;
679:     temp = find(name,Msg.command->list);
680:     return getValue(req.data, temp, IsValid);
681: }
682:
683: unsigned char MessageManipulator::getUINT08Value(char* name,SDMSerreqst& req, bool*
IsValid)
684: {
685:     value_t temp;
686:     temp = getValue(name,req,IsValid);
687:     temp = cast(temp,_UINT08);
688:     return temp.value.uint08;
689: }
690:
691: signed char MessageManipulator::getINT08Value(char* name,SDMSerreqst& req, bool* IsValid)
692: {
693:     value_t temp;
694:     temp = getValue(name,req,IsValid);
695:     temp = cast(temp,_INT08);
696:     return temp.value.int08;
697: }
698:
699: unsigned short MessageManipulator::getUINT16Value(char* name,SDMSerreqst& req, bool*
IsValid)
700: {
701:     value_t temp;
702:     temp = getValue(name,req,IsValid);
703:     temp = cast(temp,_UINT16);
704:     return temp.value.uint16;
705: }
706:
707: short MessageManipulator::getINT16Value(char* name,SDMSerreqst& req, bool* IsValid)
708: {
709:     value_t temp;
710:     temp = getValue(name,req,IsValid);
711:     temp = cast(temp,_INT16);
712:     return temp.value.int16;
713: }
714:
715: unsigned long MessageManipulator::getUINT32Value(char* name,SDMSerreqst& req, bool*
IsValid)

```

```

716: {
717:     value_t temp;
718:     temp = getValue(name, req, IsValid);
719:     temp = cast(temp, _UINT32);
720:     return temp.value.uint32;
721: }
722:
723: long MessageManipulator::getINT32Value(char* name, SDMSerreqst& req, bool* IsValid)
724: {
725:     value_t temp;
726:     temp = getValue(name, req, IsValid);
727:     temp = cast(temp, _INT32);
728:     return temp.value.int32;
729: }
730:
731: float MessageManipulator::getFLOAT32Value(char* name, SDMSerreqst& req, bool* IsValid)
732: {
733:     value_t temp;
734:     temp = getValue(name, req, IsValid);
735:     temp = cast(temp, _FLOAT32);
736:     return temp.value.float32;
737: }
738:
739: double MessageManipulator::getFLOAT64Value(char* name, SDMSerreqst& req, bool* IsValid)
740: {
741:     value_t temp;
742:     temp = getValue(name, req, IsValid);
743:     temp = cast(temp, _FLOAT64);
744:     return temp.value.float64;
745: }
746:
747: bool MessageManipulator::setValue(char* name, SDMService& request, value_t val)
748: {
749:     var* temp;
750:     temp = find(name, Msg.command->list);
751:     return setValue(request.data, temp, val);
752: }
753:
754: //////////////////////////////////////
755: //
756: //      SET FUNCTIONS

```

```

757: //
758: //////////////////////////////////////
759:
760: bool MessageManipulator::setArray(char* name,SDMData& dat,enum msg_type type, const void*
buffer,int length)
761: {
762:     var* temp;
763:
764:     if (type == DATAMSG)
765:     {
766:         if (Msg.data == NULL) return false;
767:         temp = find (name,Msg.data->list);
768:     }
769:     else if (type == FAULTMSG)
770:     {
771:         if (Msg.fault == NULL) return false;
772:         temp = find (name,Msg.fault->list);
773:     }
774:     else//COMMANDMSG is an error
775:         return false;
776:
777:     if (temp != NULL)
778:     {
779:         int MinLength = length < temp->length ? length : temp->length;
780:         if (MinLength <= temp->length)
781:             memcpy(dat.msg + temp->start, buffer, MinLength);
782:         return true;
783:     }
784:
785:     return false;
786: }
787:
788: bool MessageManipulator::setArray(char* name,SDMService& ser, const void* buffer,int length)
789: {
790:     var* temp;
791:
792:     if (Msg.command == NULL) return false;
793:     temp = find (name,Msg.command->list);
794:
795:     if (temp != NULL)
796:     {

```

```

797:     int MinLength = length < temp->length ? length : temp->length;
798:     if (MinLength <= temp->length)
799:         memcpy(ser.data + temp->start, buffer, MinLength);
800:     return true;
801: }
802: return false;
803: }
804:
805: bool MessageManipulator::setArray(char* name,SDMCommand& cmd, const void* buffer,int
length)
806: {
807:     return setArray(name,static_cast<SDMService&>(cmd),buffer,length);
808: }
809:
810: ////////////////////////////////// Sets for SDMService //////////////////////////////////
811:
812: bool MessageManipulator::setValue(char* name,SDMService& request,unsigned char val)
813: {
814:     if(Msg.type != REQUEST)
815:         return false;
816:     value_t temp;
817:     temp.value.uint08 = val;
818:     temp.type = _UINT08;
819:     return setValue(name,request,temp);
820: }
821:
822: bool MessageManipulator::setValue(char* name,SDMService& request,char val)
823: {
824:     if(Msg.type != REQUEST)
825:         return false;
826:     value_t temp;
827:     temp.value.int08 = val;
828:     temp.type = _INT08;
829:     return setValue(name,request,temp);
830: }
831:
832: bool MessageManipulator::setValue(char* name,SDMService& request,unsigned short val)
833: {
834:     if(Msg.type != REQUEST)
835:         return false;
836:     value_t temp;

```

```

837:     temp.value.uint16 = val;
838:     temp.type = _UINT16;
839:     return setValue(name,request,temp);
840: }
841:
842: bool MessageManipulator::setValue(char* name,SDMService& request,short val)
843: {
844:     if(Msg.type != REQUEST)
845:         return false;
846:     value_t temp;
847:     temp.value.int16 = val;
848:     temp.type = _INT16;
849:     return setValue(name,request,temp);
850: }
851:
852: bool MessageManipulator::setValue(char* name,SDMService& request,unsigned long val)
853: {
854:     if(Msg.type != REQUEST)
855:         return false;
856:     value_t temp;
857:     temp.value.uint32 = val;
858:     temp.type = _UINT32;
859:     return setValue(name,request,temp);
860: }
861:
862: bool MessageManipulator::setValue(char* name,SDMService& request,long val)
863: {
864:     if(Msg.type != REQUEST)
865:         return false;
866:     value_t temp;
867:     temp.value.int32 = val;
868:     temp.type = _INT32;
869:     return setValue(name,request,temp);
870: }
871:
872: bool MessageManipulator::setValue(char* name,SDMService& request,float val)
873: {
874:     if(Msg.type != REQUEST)
875:         return false;
876:     value_t temp;
877:     temp.value.float32 = val;

```

```

878:     temp.type = _FLOAT32;
879:     return setValue(name,request,temp);
880: }
881:
882: bool MessageManipulator::setValue(char* name,SDMService& request,double val)
883: {
884:     if(Msg.type != REQUEST)
885:         return false;
886:     value_t temp;
887:     temp.value.float64 = val;
888:     temp.type = _FLOAT64;
889:     return setValue(name,request,temp);
890: }
891:
892: bool MessageManipulator::setValue(char* Buffer, var* Variable, value_t Value)
893: {
894:     if (Buffer == NULL || Variable == NULL) return false;
895:     // cast if needed
896:     int start = Variable->start;
897:     Value = cast(Value, Variable->type);
898:     switch(Variable->type)
899:     {
900:         case _UINT08:
901:             PUT_UCHAR(&(Buffer[start]),Value.value.uint08);
902:             break;
903:         case _INT08:
904:             PUT_CHAR(&(Buffer[start]),Value.value.int08);
905:             break;
906:         case _UINT16:
907:             PUT_USHORT(&(Buffer[start]),Value.value.uint16);
908:             break;
909:         case _INT16:
910:             PUT_SHORT(&(Buffer[start]),Value.value.int16);
911:             break;
912:         case _UINT32:
913:             PUT_ULONG(&(Buffer[start]),Value.value.uint32);
914:             break;
915:         case _INT32:
916:             PUT_LONG(&(Buffer[start]),Value.value.int32);
917:             break;
918:         case _FLOAT32:

```

```

919:         PUT_FLOAT(&(Buffer[start]),Value.value.float32);
920:         break;
921:     case _FLOAT64:
922:         PUT_DOUBLE(&(Buffer[start]),Value.value.float64);
923:         break;
924:     case _EMPTY:
925:         break;
926: }
927: return true;
928: }
929:
930: bool MessageManipulator::setValue(char* name,SDMCommand& command,value_t val)
931: {
932:     var* temp;
933:     temp = find(name,Msg.command->list);
934:     return setValue(command.data, temp, val);
935: }
936:
937: /////////////// Sets for SDMCommand ///////////////
938:
939: bool MessageManipulator::setValue(char* name,SDMCommand& command,unsigned char val)
940: {
941:     if(Msg.type == NOTIFICATION)
942:         return false;
943:     value_t temp;
944:     temp.value.uint08 = val;
945:     temp.type = _UINT08;
946:     return setValue(name,command,temp);
947: }
948:
949: bool MessageManipulator::setValue(char* name,SDMCommand& command,char val)
950: {
951:     if(Msg.type == NOTIFICATION)
952:         return false;
953:     value_t temp;
954:     temp.value.int08 = val;
955:     temp.type = _INT08;
956:     return setValue(name,command,temp);
957: }
958:
959: bool MessageManipulator::setValue(char* name,SDMCommand& command,unsigned short val)

```

```

960: {
961:     if(Msg.type == NOTIFICATION)
962:         return false;
963:     value_t temp;
964:     temp.value.uint16 = val;
965:     temp.type = _UINT16;
966:     return setValue(name,command,temp);
967: }
968:
969: bool MessageManipulator::setValue(char* name,SDMCommand& command,short val)
970: {
971:     if(Msg.type == NOTIFICATION)
972:         return false;
973:     value_t temp;
974:     temp.value.int16 = val;
975:     temp.type = _INT16;
976:     return setValue(name,command,temp);
977: }
978:
979: bool MessageManipulator::setValue(char* name,SDMCommand& command,unsigned long val)
980: {
981:     if(Msg.type == NOTIFICATION)
982:         return false;
983:     value_t temp;
984:     temp.value.uint32 = val;
985:     temp.type = _UINT32;
986:     return setValue(name,command,temp);
987: }
988:
989: bool MessageManipulator::setValue(char* name,SDMCommand& command,long val)
990: {
991:     if(Msg.type == NOTIFICATION)
992:         return false;
993:     value_t temp;
994:     temp.value.int32 = val;
995:     temp.type = _INT32;
996:     return setValue(name,command,temp);
997: }
998:
999: bool MessageManipulator::setValue(char* name,SDMCommand& command,float val)
1000: {

```



```

1001:  if(Msg.type == NOTIFICATION)
1002:      return false;
1003:  value_t temp;
1004:  temp.value.float32 = val;
1005:  temp.type = _FLOAT32;
1006:  return setValue(name,command,temp);
1007: }
1008:
1009: bool MessageManipulator::setValue(char* name,SDMCommand& command,double val)
1010: {
1011:  if(Msg.type == NOTIFICATION)
1012:      return false;
1013:  value_t temp;
1014:  temp.value.float64 = val;
1015:  temp.type = _FLOAT64;
1016:  return setValue(name,command,temp);
1017: }
1018:
1019: bool MessageManipulator::setValue(char* name,SDMData& data,value_t val,enum msg_type
type)
1020: {
1021:  var* temp;
1022:
1023:  if(type == DATAMSG)
1024:  {
1025:      //Be sure a data message is defined
1026:      if (Msg.data == NULL)
1027:          return false;
1028:      else
1029:          temp = find(name,Msg.data->list);
1030:  }
1031:  else if (type == FAULTMSG)
1032:  {
1033:      //Be sure a fault message is defined
1034:      if (Msg.fault == NULL)
1035:          return false;
1036:      else
1037:          temp = find(name,Msg.fault->list);
1038:  }
1039:  else//COMMANDMSG results in an error
1040:      return false;

```

```

1041:
1042:  return setValue(data.msg, temp, val);
1043: }
1044:
1045: ///////////////   Sets for SDMDData   ///////////////
1046:
1047: bool MessageManipulator::setValue(char* name,SDMDData& data,unsigned char val,enum
msg_type type)
1048: {
1049:  value_t temp;
1050:  temp.value.uint08 = val;
1051:  temp.type = _UINT08;
1052:  return setValue(name,data,temp,type);
1053: }
1054:
1055: bool MessageManipulator::setValue(char* name,SDMDData& data,char val,enum msg_type type)
1056: {
1057:  value_t temp;
1058:  temp.value.int08 = val;
1059:  temp.type = _INT08;
1060:  return setValue(name,data,temp,type);
1061: }
1062:
1063: bool MessageManipulator::setValue(char* name,SDMDData& data,unsigned short val,enum
msg_type type)
1064: {
1065:  value_t temp;
1066:  temp.value.uint16 = val;
1067:  temp.type = _UINT16;
1068:  return setValue(name,data,temp,type);
1069: }
1070:
1071: bool MessageManipulator::setValue(char* name,SDMDData& data,short val,enum msg_type type)
1072: {
1073:  value_t temp;
1074:  temp.value.int16 = val;
1075:  temp.type = _INT16;
1076:  return setValue(name,data,temp,type);
1077: }
1078:
1079: bool MessageManipulator::setValue(char* name,SDMDData& data,unsigned long val,enum
msg_type type)

```

```

1080: {
1081:     value_t temp;
1082:     temp.value.uint32 = val;
1083:     temp.type = _UINT32;
1084:     return setValue(name,data,temp,type);
1085: }
1086:
1087: bool MessageManipulator::setValue(char* name,SDMData& data,long val,enum msg_type type)
1088: {
1089:     value_t temp;
1090:     temp.value.int32 = val;
1091:     temp.type = _INT32;
1092:     return setValue(name,data,temp,type);
1093: }
1094:
1095: bool MessageManipulator::setValue(char* name,SDMData& data,float val,enum msg_type type)
1096: {
1097:     value_t temp;
1098:     temp.value.float32 = val;
1099:     temp.type = _FLOAT32;
1100:     return setValue(name,data,temp,type);
1101: }
1102:
1103: bool MessageManipulator::setValue(char* name,SDMData& data,double val,enum msg_type type)
1104: {
1105:     value_t temp;
1106:     temp.value.float64 = val;
1107:     temp.type = _FLOAT64;
1108:     return setValue(name,data,temp,type);
1109: }
1110:
1111: SDMMMessage_ID MessageManipulator::getMsgID(enum msg_type type)
1112: {
1113:     SDMMMessage_ID Result('\0','\0');
1114:     switch(type)
1115:     {
1116:         case COMMANDMSG:
1117:             if(Msg.command != NULL)
1118:                 Result = Msg.command->id;
1119:             return Result;
1120:             break;

```

```
1121:     case DATAMSG:
1122:         if(Msg.data != NULL)
1123:             Result = Msg.data->id;
1124:         return Result;
1125:         break;
1126:     case FAULTMSG:
1127:         if(Msg.fault != NULL)
1128:             Result = Msg.fault->id;
1129:         return Result;
1130:         break;
1131: }
1132: return Result;
1133: }
1134:
1135: void MessageManipulator::applyGetScaleFactor(bool apply)
1136: {
1137:     scaled = apply;
1138: }
```

File: sdm/common/MessageLogger/SDMMessageLogger.h

```
1: //This is a wrapper class for the MessageLogger class. The SDMMessageLogger is to be
2: //used in the SDMmessage class as a static member which logs all messages sent and
3: //received transparently as the message classes call Send() and Unmarshal().
4:
5: #ifndef _SDM_MESSAGE_LOGGER_H_
6: #define _SDM_MESSAGE_LOGGER_H_
7: #include "MessageLogger.h"
8:
9: class SDMmessage;
10:
11: class SDMLIB_API SDMMessageLogger
12: {
13: public:
14: SDMMessageLogger();
15: bool MessageReceived(const SDMmessage &Message)
16:     { return Logger.MessageReceived(&Message); }
17: bool MessageSent(const SDMmessage &Message)
18:     { return Logger.MessageSent(&Message); }
19: private:
20: MessageLogger Logger;
21: };
22:
23: #endif
```

File: sdm/common/MessageLogger/Makefile

```
1: #MessageLogger makefile
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: .PHONY: all clean distclean
7:
8: all: MessageLogger.o SDMMMessageLogger.o
9:
10: MessageLogger.o: MessageLogger.cpp MessageLogger.h
11: $(CXX) $(CXXFLAGS) $(MESSAGELOGGINGFLAGS) -fPIC -c $<
12:
13: SDMMMessageLogger.o: SDMMMessageLogger.cpp SDMMMessageLogger.h
14: $(CXX) $(CXXFLAGS) $(MESSAGELOGGINGFLAGS) -fPIC -c $<
15:
16: clean:
17: rm -f *.o *~
18:
19: distclean: clean
```

File: sdm/common/MessageLogger/MessageLogger.cpp

```
1: #include <stdlib.h>
2: #include <string.h>
3: #include <sys/types.h>
4: #include <sys/stat.h>
5: #include <fcntl.h>
6: #include <stdio.h>
7: #include <errno.h>
8:
9: #include <unistd.h>      /*Needed to handle open() and write() file I/O*/
10:
11: #include "MessageLogger.h"
12: #include "../message_defs.h"
13: #include "../message/SDMxTEDS.h"
14: #include "../message/SDMCancelxTEDS.h"
15: #include "../message/SDMSubreqst.h"
16: #include "../message/SDMDeletesub.h"
17: #include "../message/SDMData.h"
18: #include "../message/SDMReqCode.h"
19: #include "../message/SDMConsume.h"
20: #include "../message/SDMReqReg.h"
21: #include "../message/SDMRegInfo.h"
22: #include "../message/SDMRegPM.h"
23: #include "../message/SDMCancel.h"
24: #include "../message/SDMService.h"
25: #include "../message/SDMSerreqst.h"
26: #include "../message/SDMCommand.h"
27: #include "../message/SDMReqxTEDS.h"
28: #include "../message/SDMxTEDSInfo.h"
29: #include "../message/SDMError.h"
30: #include "../message/SDMReady.h"
31: #include "../message/SDMTat.h"
32: #include "../message/SDMTask.h"
33: #include "../message/SDMTaskFinished.h"
34: #include "../message/SDMSearch.h"
35: #include "../message/SDMSearchReply.h"
36: #include "../message/SDMHeartbeat.h"
37: #include "../message/SDMDMLLeader.h"
38: #include "../message/SDMVarReq.h"
39: #include "../message/SDMTaskError.h"
```

```

40: #include "../message/SDMVarInfo.h"
41:
42: MessageLogger::MessageLogger():numTypes(0), fd(-1), init_done(false), LogIdListHead(NULL)
43: {
44: }
45: MessageLogger::~MessageLogger()
46: {
47: if (fd > 0)
48:     close(fd);
49: }
50:
51: bool MessageLogger::AddMessageType(const SDMCommand *msg)
52: {
53: unsigned char msg_type;
54: LogID nextID;
55: SDMComponent_ID id;
56:
57: msg_type = GET_UCHAR(&msg->data[0]);
58: id.Unmarshal(msg->data, 1);
59:
60: for(LogID* cur = LogIdListHead; cur != NULL; cur = cur->next)
61: {
62:     if(cur->msg_type == msg_type && cur->component == id)
63:     {
64:         return false;
65:     }
66: }
67: nextID.msg_type = msg_type;
68: nextID.component = id;
69: nextID.next = LogIdListHead;
70: LogIdListHead = &nextID;
71: numTypes++;
72: return true;
73: }
74:
75: bool MessageLogger::RemoveMessageType(const SDMCommand *msg)
76: {
77: unsigned char msg_type;
78: SDMComponent_ID id;
79: bool ignore_sid = false;
80: LogID* prevPtr = NULL;

```



```

81: LogID* cur;
82:
83: msg_type = GET_UCHAR(&msg->data[0]);
84: id.Unmarshal(msg->data, 1);
85:
86: if (id.getAddress() == 0 && id.getPort() == 0 && id.getSensorID() == 0 && msg_type == ' \0')
    //Remove all messages
87: {
88:     LogIdListHead = NULL;
89:     numTypes = 0;
90:     return true;
91: }
92: else if (id.getAddress() == 0 && id.getPort() == 0 && id.getSensorID() == 0)
93: {
94:     ignore_sid = true;
95: }
96:
97: for(cur = LogIdListHead; cur != NULL; cur = cur->next)
98: {
99:     if (cur->msg_type == msg_type && (ignore_sid || cur->component == id)) //Find the item to
remove
100:     {
101:         break;
102:     }
103:     prevPtr = cur;
104: }
105:
106: if (cur == NULL) //Item not found
107: {
108:     return false;
109: }
110: else
111: {
112:     if(prevPtr != NULL)
113:     {
114:         prevPtr->next = cur->next;
115:     }
116:     else
117:     {
118:         LogIdListHead = cur->next;
119:     }

```

```

120:     }
121:     numTypes--;
122:     return true;
123: }
124:
125: bool MessageLogger::Contains(char msg_type)
126: {
127:     for(LogID* cur = LogIdListHead; cur != NULL; cur = cur->next)
128:     {
129:         if (cur->msg_type == msg_type || cur->msg_type == '\0')
130:             return true;
131:     }
132:     return false;
133: }
134:
135: bool MessageLogger::Contains(char msg_type, SDMComponent_ID id)
136: {
137:     bool match_sensorid = true;
138:     bool match_ipaddr = true;
139:     bool match_port = true;
140:     for(LogID* cur = LogIdListHead; cur != NULL; cur = cur->next)
141:     {
142:         //Test which parts of the component ID should be matched
143:         if (cur->component.getSensorID() == 0)
144:             match_sensorid = false;
145:         if (cur->component.getAddress() == 0)
146:             match_ipaddr = false;
147:         if (cur->component.getPort() == 0)
148:             match_port = false;
149:
150:         if (cur->msg_type == msg_type || cur->msg_type == '\0')
151:             if (!match_sensorid || (cur->component.getSensorID() == id.getSensorID()))
152:                 if (!match_ipaddr || (cur->component.getAddress() == id.getAddress()))
153:                     if (!match_port || (cur->component.getPort() == id.getPort()))
154:                         return true;
155:
156:     }
157:     return false;
158: }
159:
160: bool MessageLogger::Contains(char msg_type, SDMComponent_ID id1, SDMComponent_ID id2)

```

```

161: {
162:     bool match_sensorid = true;
163:     bool match_ipaddr = true;
164:     bool match_port = true;
165:     for(LogID* cur = LogIdListHead; cur != NULL; cur = cur->next)
166:     {
167:         //Test which parts of the component ID should be matched
168:         if (cur->component.getSensorID() == 0)
169:             match_sensorid = false;
170:         if (cur->component.getAddress() == 0)
171:             match_ipaddr = false;
172:         if (cur->component.getPort() == 0)
173:             match_port = false;
174:
175:         if (cur->msg_type == msg_type || cur->msg_type == '\0')
176:             if (!match_sensorid || (cur->component.getSensorID() == id1.getSensorID()) || (cur-
177: >component.getSensorID() == id2.getSensorID()))
178:                 if (!match_ipaddr || (cur->component.getAddress() == id1.getAddress()) || (cur-
179: >component.getAddress() == id2.getAddress()))
180:                     if (!match_port || (cur->component.getPort() == id1.getPort()) || (cur-
181: >component.getPort() == id2.getPort()))
182:                         return true;
183:     }
184:     return false;
185: }
186:
187: bool MessageLogger::IsEmpty()
188: {
189:     return (numTypes == 0);
190: }
191:
192: bool MessageLogger::MessageReceived(const SDMmessage *msg)
193: {
194:     char msg_type = msg->GetMsgName();
195:     char message_buf[3*BUFSIZE];
196:     char comp_id_buf[22];
197:     SDMComponent_ID compid1;
198:     SDMComponent_ID compid2;
199:     int num_ids = 0;
200:     bool contains = false;

```

```

200:   GetCompIDs(msg, comp_id_buf, num_ids);
201:
202:   if (fd < 0)
203:       return false;
204:   if (num_ids == 0)
205:       contains = this->Contains(msg_type);
206:   else if (num_ids == 1)
207:   {
208:       compid1.Unmarshal(comp_id_buf, 0);
209:       contains = this->Contains(msg_type, compid1);
210:   }
211:   else if (num_ids == 2)
212:   {
213:       compid1.Unmarshal(comp_id_buf, 0);
214:       compid2.Unmarshal(comp_id_buf, HEADER_SIZE);
215:       contains = this->Contains(msg_type, compid1, compid2);
216:   }
217:
218:   if (contains)
219:   {
220:       msg->MsgToString(message_buf, sizeof(message_buf));
221:       return WriteLogMessage("IN  ", message_buf);
222:   }
223:
224:   return false;
225: }
226:
227: bool MessageLogger::MessageSent(const SDMmessage *msg)
228: {
229:     unsigned char msg_type = msg->GetMsgName();
230:     char message_buf[3*BUFSIZE];
231:     char comp_id_buf[22];
232:     SDMComponent_ID compid1;
233:     SDMComponent_ID compid2;
234:     int num_ids = 0;
235:     bool contains = false;
236:
237:     GetCompIDs(msg, comp_id_buf, num_ids);
238:
239:     if (fd < 0)
240:         return false;

```

```

241:  if (num_ids == 0)
242:      contains = this->Contains(msg_type);
243:  else if (num_ids == 1)
244:  {
245:      compid1.Unmarshal(comp_id_buf, 0);
246:      contains = this->Contains(msg_type, compid1);
247:  }
248:  else if (num_ids == 2)
249:  {
250:      compid1.Unmarshal(comp_id_buf, 0);
251:      compid2.Unmarshal(comp_id_buf, HEADER_SIZE);
252:      contains = this->Contains(msg_type, compid1, compid2);
253:  }
254:
255:  if (contains)
256:  {
257:      msg->MsgToString(message_buf, sizeof(message_buf));
258:      return WriteLogMessage("OUT ", message_buf);
259:  }
260:
261:  return false;
262: }
263:
264: void MessageLogger::SetLogFile(const char *header, const char *filename)
265: {
266:     int length;
267:
268: #ifndef WIN32
269:     fd = open(filename, O_TRUNC | O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
270: #else
271:     fd = open(filename, O_TRUNC | O_CREAT | O_RDWR);
272: #endif
273:
274:     if (fd < 0)
275:         perror("Error occurred while opening logfile:");
276:     else
277:     {
278:         length = write(fd, header, strlen(header));
279:         if (length < 0)
280:             perror("Write failed: ");
281:         init_done = true;

```

```

282:     }
283: }
284:
285: bool MessageLogger::WriteLogMessage(char *prefix, char *msg_buf)
286: {
287:     int length;
288:     char newline = '\n';
289:
290:     if (fd < 0)
291:         return false;
292:     write(fd, prefix, strlen(prefix));
293:     length = write(fd, msg_buf, strlen(msg_buf));
294:     write(fd, &newline, 1);
295:
296:     if (length < 0)
297:         return false;
298:     else
299:         return true;
300: }
301:
302: void MessageLogger::GetCompIDs(const SDMmessage *msg, char *buf, int &num_ids)
303: {
304:     unsigned char msg_type = msg->GetMsgName();
305:     SDMComponent_ID id1;
306:     SDMComponent_ID id2;
307:
308:     switch(msg_type)
309:     {
310:         case SDM_ACK:
311:         {
312:             num_ids = 0;
313:             break;
314:         }
315:         case SDM_Cancel:
316:         {
317:             SDMCancel *message = (SDMCancel*) msg;
318:             num_ids = 2;
319:             id1 = message->source;
320:             id2 = message->destination;
321:             break;
322:         }

```

```

323:     case SDM_CancelxTEDS:
324:     {
325:         SDMCancelxTEDS *message = (SDMCancelxTEDS*) msg;
326:         num_ids = 1;
327:         id1 = message->source;
328:         break;
329:     }
330:     case SDM_Code:
331:     {
332:         num_ids = 0;
333:         break;
334:     }
335:     case SDM_Command:
336:     {
337:         SDMCommand *message = (SDMCommand*) msg;
338:         num_ids = 1;
339:         id1 = message->source;
340:         break;
341:     }
342:
343:     case SDM_Consume:
344:     {
345:         SDMConsume*message = (SDMConsume*) msg;
346:         num_ids = 2;
347:         id1 = message->source;
348:         id2 = message->destination;
349:         break;
350:     }
351:     case SDM_Deletesub:
352:     {
353:         SDMDDeletesub *message = (SDMDDeletesub*) msg;
354:         num_ids = 2;
355:         id1 = message->source;
356:         id2 = message->destination;
357:         break;
358:     }
359:     case SDM_Data:
360:     {
361:         SDMDData *message = (SDMDData*) msg;
362:         num_ids = 1;
363:         id1 = message->source;

```

```

364:         break;
365:     }
366:     case SDM_DMLeader:
367:     {
368:         SDMDMLeader *message = (SDMDMLeader*)msg;
369:         num_ids = 1;
370:         id1 = message->source;
371:         break;
372:     }
373:     case SDM_Election:
374:     {
375:         num_ids = 0;
376:         break;
377:     }
378:     case SDM_Error:
379:     {
380:         SDLError *message = (SDLError*) msg;
381:         num_ids = 1;
382:         id1 = message->source;
383:         break;
384:     }
385:     case SDM_Heartbeat:
386:     {
387:         SDMHeartbeat *message = (SDMHeartbeat*)msg;
388:         num_ids = 1;
389:         id1 = message->source;
390:         break;
391:     }
392:     case SDM_Kill:
393:     {
394:         num_ids = 0;
395:         break;
396:     }
397:     case SDM_PostTask:
398:     {
399:         num_ids = 0;
400:         break;
401:     }
402:     case SDM_Ready:
403:     {
404:         SDMReady *message = (SDMReady*) msg;

```



```

405:         num_ids = 2;
406:         id1 = message->source;
407:         id2 = message->destination;
408:         break;
409:     }
410: case SDM_Search:
411:     {
412:         SDMSearch *message = (SDMSearch*) msg;
413:         num_ids = 2;
414:         id1 = message->source;
415:         id2 = message->destination;
416:         break;
417:     }
418: case SDM_SearchReply:
419:     {
420:         SDMSearchReply *message = (SDMSearchReply*)msg;
421:         num_ids = 1;
422:         id1 = message->source;
423:         break;
424:     }
425: case SDM_RegInfo:
426:     {
427:         SDMRegInfo *message = (SDMRegInfo*) msg;
428:         num_ids = 1;
429:         id1 = message->source;
430:         break;
431:     }
432: case SDM_RegPM:
433:     {
434:         SDMRegPM *message = (SDMRegPM*) msg;
435:         num_ids = 1;
436:         id1 = message->source;
437:         break;
438:     }
439: case SDM_ReqCode:
440:     {
441:         SDMReqCode *message = (SDMReqCode*) msg;
442:         num_ids = 1;
443:         id1 = message->source;
444:         break;
445:     }

```

```

446:     case SDM_ReqReg:
447:     {
448:         SDMReqReg *message = (SDMReqReg*) msg;
449:         num_ids = 2;
450:         id1 = message->source;
451:         id2 = message->destination;
452:         break;
453:     }
454:     case SDM_ReqxTEDS:
455:     {
456:         SDMReqxTEDS *message = (SDMReqxTEDS*) msg;
457:         num_ids = 2;
458:         id1 = message->source;
459:         id2 = message->destination;
460:         break;
461:     }
462:     case SDM_Serreqst:
463:     {
464:         SDMSerreqst *message = (SDMSerreqst*) msg;
465:         num_ids = 2;
466:         id1 = message->source;
467:         id2 = message->destination;
468:         break;
469:     }
470:     case SDM_Service:
471:     {
472:         SDMService *message = (SDMService *) msg;
473:         num_ids = 2;
474:         id1 = message->source;
475:         id2 = message->destination;
476:         break;
477:     }
478:     case SDM_Subreqst:
479:     {
480:         SDMSubreqst *message = (SDMSubreqst*) msg;
481:         num_ids = 2;
482:         id1 = message->source;
483:         id2 = message->destination;
484:         break;
485:     }
486:     case SDM_Task:

```

```

487:     {
488:         SDMTask *message = (SDMTask*) msg;
489:         num_ids = 1;
490:         id1 = message->source;
491:         break;
492:     }
493: case SDM_TaskError:
494:     {
495:         SDMTaskError *message = (SDMTaskError*) msg;
496:         num_ids = 1;
497:         id1 = message->source;
498:         break;
499:     }
500: case SDM_TaskFinished:
501:     {
502:         SDMTaskFinished *message = (SDMTaskFinished*) msg;
503:         num_ids = 1;
504:         id1 = message->source;
505:         break;
506:     }
507: case SDM_Tat:
508:     {
509:         SDMTat *message = (SDMTat*) msg;
510:         num_ids = 1;
511:         id1 = message->destination;
512:         break;
513:     }
514: case SDM_VarInfo:
515:     {
516:         SDMVarInfo *message = (SDMVarInfo*) msg;
517:         num_ids = 1;
518:         id1 = message->source;
519:         break;
520:     }
521: case SDM_VarReq:
522:     {
523:         SDMVarReq *message = (SDMVarReq*) msg;
524:         num_ids = 1;
525:         id1 = message->source;
526:         break;
527:     }

```

```

528:     case SDM_xTEDSInfo:
529:     {
530:         SDMxTEDSInfo *message = (SDMxTEDSInfo*) msg;
531:         num_ids = 1;
532:         id1 = message->source;
533:         break;
534:     }
535:     case SDM_xTEDS:
536:     {   SDMxTEDS *message = (SDMxTEDS*) msg;
537:         num_ids = 1;
538:         id1 = message->source;
539:         break;
540:     }
541: }
542: if (num_ids == 1)
543: {
544:     id1.Marshal(buf, 0);
545: }
546: else if (num_ids == 2)
547: {
548:     id1.Marshal(buf, 0);
549:     id2.Marshal(buf, HEADER_SIZE);
550: }
551: }
552: bool MessageLogger::NeedsInit()
553: {
554:     return !init_done;
555: }
556:
557: void MessageLogger::LogAllMessageTypes()
558: {
559:     unsigned char msg_type = '\0';
560:     LogID nextID;
561:     SDMComponent_ID id;
562:     //Make sure this operation hasn't already been called
563:     for(LogID* cur = LogIdListHead; cur != NULL; cur = cur->next)
564:     {
565:         if(cur->msg_type == msg_type && cur->component == id)
566:             return;
567:     }
568:     nextID.msg_type = msg_type;

```

```
569:  nextID.component = id;
570:  nextID.next = LogIdListHead;
571:  LogIdListHead = &nextID;
572:  numTypes++;
573: }
```

File: sdm/common/MessageLogger/MessageLogger.h

```
1: #ifndef __MESSAGE_LOGGER_H_
2: #define __MESSAGE_LOGGER_H_
3:
4: #ifndef BUILD_WITH_MESSAGE_LOGGING
5: # include "../message/SDMCommand.h"
6: # include "../message/SDMmessage.h"
7: #endif
8: #include "../message/SDMComponent_ID.h"
9:
10:
11: #ifdef BUILD_WITH_MESSAGE_LOGGING
12: class SDMCommand;
13: class SDMmessage;
14: #endif
15:
16: struct LogID
17: {
18:     SDMComponent_ID component;
19:     unsigned char msg_type;
20:     LogID* next;
21:     LogID():component(),msg_type(0)
22: { }
23: };
24:
25: class SDMLIB_API MessageLogger
26: {
27: public:
28:     MessageLogger();
29:     ~MessageLogger();
30:     bool AddMessageType(const SDMCommand *msg);    //Adds message types for logging
31:     bool RemoveMessageType(const SDMCommand *msg); //Removes message types for logging
32:     bool MessageReceived(const SDMmessage *msg);  //Called when message is received
33:     bool MessageSent(const SDMmessage *msg);     //Called when message is sent
34:     bool Contains(char msg_type);                //Determines whether a type is being logged
35:     bool Contains(char msg_type, SDMComponent_ID id);
36:     bool Contains(char msg_type, SDMComponent_ID id1, SDMComponent_ID id2);
37:     void SetLogFile(const char *header, const char *filename);
38:     bool IsEmpty();                             //Returns whether message types are being logged
39:     bool NeedsInit();                            //Used to determine if log file needs to be set
```

```
40: void LogAllMessageTypes();          //Tell the service to log everything
41: private:
42: bool WriteLogMessage(char *prefix, char *msg_buf);
43: void GetCompIDs(const SDMmessage *msg, char *buf, int &num_ids);
44:
45: int numTypes;          //A counter of the number of types being logged
46: int fd;               //File descriptor of the log file
47: bool init_done;       //Flag to determine whether log file needs to be set
48: LogID* LogIdListHead;
49: };
50:
51:
52: #endif
```

File: sdm/common/MessageLogger/SDMMessageLogger.cpp

```
1: #include <sys/types.h>
2: #include <unistd.h>
3: #include <stdio.h>
4: #include "SDMMessageLogger.h"
5:
6: /*
7:  * Default constructor, set the log file to a default name and to log all messages.
8:  *
9:  */
10: SDMMessageLogger::SDMMessageLogger():Logger()
11: {
12: #ifdef WIN32
13: //
14: // WINDOWS
15: //
16: //Get the process pid
17: int pid = GetCurrentProcessId();
18: //Create a unique filename
19: char Filename[64];
20: sprintf(Filename, "SDMMessages%d.log",pid);
21: Logger.SetLogFile("-----Log of SDM Messages Sent and Received----- \n" \
22: "-----", Filename);
23: Logger.LogAllMessageTypes();
24: #else
25: //
26: // LINUX
27: //
28: //Get the process pid
29: int pid = getpid();
30: //Create a unique filename
31: char Filename[64];
32: sprintf(Filename, "SDMMessages%d.log",pid);
33: Logger.SetLogFile("-----Log of SDM Messages Sent and Received----- \n" \
34: "-----", Filename);
35: Logger.LogAllMessageTypes();
36: #endif
37: }
```


File: sdm/common/checksum/crctable.c

```
1: /*****
2: /*          Start of crctable.c          */
3: /*****
4: /*
5: /* Author : Ross Williams (ross@guest.adelaide.edu.au).    */
6: /* Date   : 3 June 1993.          */
7: /* Version : 1.0.          */
8: /* Status  : Public domain.          */
9: /*
10: /* Description : This program writes a CRC lookup table (suitable for
11: /* inclusion in a C program) to a designated output file. The program can be*/
12: /* statically configured to produce any table covered by the Rocksoft^tm */
13: /* Model CRC Algorithm. For more information on the Rocksoft^tm Model CRC */
14: /* Algorithm, see the document titled "A Painless Guide to CRC Error */
15: /* Detection Algorithms" by Ross Williams (ross@guest.adelaide.edu.au.). */
16: /* This document is likely to be in "ftp.adelaide.edu.au/pub/rocksoft". */
17: /*
18: /* Note: Rocksoft is a trademark of Rocksoft Pty Ltd, Adelaide, Australia. */
19:
20: /*
21: /*****
22:
23: #include <stdio.h>
24: #include <stdlib.h>
25: #include "crcmodel.h"
26:
27: /*****
28:
29: /* TABLE PARAMETERS          */
30: /* =====          */
31: /* The following parameters entirely determine the table to be generated. */
32: /* You should need to modify only the definitions in this section before */
33: /* running this program.          */
34: /*
35: /* TB_FILE is the name of the output file.          */
36: /* TB_WIDTH is the table width in bytes (either 2 or 4).          */
37: /* TB_POLY is the "polynomial", which must be TB_WIDTH bytes wide. */
38:
39: /* TB_REVER indicates whether the table is to be reversed (reflected). */
```

```

40: /* */
41: /* Example: */
42: /* */
43: /* #define TB_FILE "crctable.out" */
44: /* #define TB_WIDTH 2 */
45: /* #define TB_POLY 0x8005L */
46: /* #define TB_REVER TRUE */
47:
48:
49: #define TB_FILE "crctable.out"
50: #define TB_WIDTH 4
51: #define TB_POLY 0x04C11DB7L
52: #define TB_REVER TRUE
53:
54: /*****
55: /* Miscellaneous definitions. */
56:
57: #define LOCAL static
58:
59: FILE *outfile;
60: #define WR(X) fprintf(outfile,(X))
61: #define WP(X,Y) fprintf(outfile,(X),(Y))
62:
63: /*****
64:
65: LOCAL void chk_err P_((char *));
66: LOCAL void chk_err (mess)
67:
68:
69: /* If mess is non-empty, write it out and abort. Otherwise, check the error */
70: /* status of outfile and abort if an error has occurred. */
71:
72: char *mess;
73: {
74: if (mess[0] != 0 ) {printf("%s \n",mess); exit(EXIT_FAILURE);}
75: if (ferror(outfile)) {perror("chk_err"); exit(EXIT_FAILURE);}
76: }
77:
78:
79: /*****
80:

```

```

81: LOCAL void chkparam P_((void));
82: LOCAL void chkparam ()
83: {
84: if ((TB_WIDTH != 2) && (TB_WIDTH != 4))
85:
86:   chk_err("chkparam: Width parameter is illegal.");
87: if ((TB_WIDTH == 2) && (TB_POLY & 0xFFFF0000L))
88:   chk_err("chkparam: Poly parameter is too wide.");
89: if ((TB_REVER != FALSE) && (TB_REVER != TRUE))
90:
91:   chk_err("chkparam: Reverse parameter is not boolean.");
92: }
93:
94: /*****
95:
96: LOCAL void gentable P_((void));
97: LOCAL void gentable ()
98: {
99:
100: WR("*****/\n");
101: WR("/*
102: WR("/* CRC LOOKUP TABLE
103: WR("/* =====
104: WR("/* The following CRC lookup table was generated automatically */\n");
105: WR("/* by the Rocksoft^tm Model CRC Algorithm Table Generation */\n");
106:
107: WR("/* Program V1.0 using the following model parameters: */\n");
108: WR("/*
109: WP("/* Width : %1lu bytes. */\n",
110:
111: (ulong) TB_WIDTH);
112: if (TB_WIDTH == 2)
113:
114: WP("/* Poly : 0x%04lX */\n",
115: (ulong) TB_POLY);
116: else
117: WP("/* Poly : 0x%08lXL */\n",
118: (ulong) TB_POLY);
119: if (TB_REVER)
120:
121: WR("/* Reverse : TRUE. */\n");

```

```

122: else
123: WR("/* Reverse : FALSE.                */\n");
124: WR("/*                                */\n");
125: WR("/* For more information on the Rocksoft^tm Model CRC Algorithm, */\n");
126: WR("/* see the document titled \"A Painless Guide to CRC Error    */\n");
127:
128: WR("/* Detection Algorithms \" by Ross Williams                */\n");
129: WR("/* (ross@guest.adelaide.edu.au.). This document is likely to be */\n");
130: WR("/* in the FTP archive \"ftp.adelaide.edu.au/pub/rocksoft \" .  */\n");
131:
132: WR("/*                                */\n");
133: WR("/******\n");
134:
135: WR("\n");
136: switch (TB_WIDTH)
137: {
138: case 2: WR("unsigned short crctable[256] = \n{ \n"); break;
139: case 4: WR("unsigned long  crctable[256] = \n{ \n"); break;
140: default: chk_err("gentable: TB_WIDTH is invalid.");
141:
142: }
143: chk_err("");
144:
145: {
146: int i;
147: cm_t cm;
148:
149: char *form  = (TB_WIDTH==2) ? "0x%04lX" : "0x%08lXL";
150: int  perline = (TB_WIDTH==2) ? 8 : 4;
151:
152: cm.cm_width = TB_WIDTH*8;
153: cm.cm_poly  = TB_POLY;
154: cm.cm_refin = TB_REVER;
155:
156:
157: for (i=0; i<256; i++)
158: {
159: WR(" ");
160: WP(form,(ulong) cm_tab(&cm,i));
161: if (i != 255) WR(",");
162:

```

```

163:   if (((i+1) % perline) == 0) WR(" \n");
164:   chk_err("");
165:   }
166:
167: WR("}; \n");
168: WR(" \n");
169:
170: WR("/******\n");
171: WR("/*      End of CRC Lookup Table      */\n");
172: WR("/******\n");
173: chk_err("");
174: }
175:
176: }
177:
178: /*****
179:
180: int main ()
181: {
182: printf(" \n");
183: printf("Rocksoft^tm Model CRC Algorithm Table Generation Program V1.0 \n");
184:
185: printf("----- \n");
186: printf("Output file is \"%s \". \n",TB_FILE);
187: chkparam();
188: outfile = fopen(TB_FILE,"w"); chk_err("");
189: gentable();
190: if (fclose(outfile) != 0)
191:
192:   chk_err("main: Couldn't close output file.");
193: printf(" \nSUCCESS: The table has been successfully written. \n");
194: return 0;
195: }
196:
197: /*****
198: /*      End of crcstable.c      */
199: /*****
200:
201: /*****

```

File: sdm/common/checksum/checksum.c

```
1: /*calculates a check sum for SDM messages
2: inspired by "A Painless Guide to CRC Error Detection Algorithms
3: at http://www.geocities.com/SiliconValley/Pines/8659/crc/htm
4: accessed 5APR2005*/
5:
6: #include "crcmodel.h"
7: #include <string.h>
8: #include <stdio.h>
9:
10: /*return the checksum of given message*/
11: int checksum(char* msg,int len)
12: {
13: int i;
14: cm_t model;
15: p_cm_t p_cm;
16: p_cm = &model;
17:
18: /*CRC-32*/
19: p_cm->cm_width = 32;
20: p_cm->cm_poly = 0x04C11DB7;
21: p_cm->cm_init = 0xFFFFFFFF;
22: p_cm->cm_refin = TRUE;
23: p_cm->cm_refot = TRUE;
24: p_cm->cm_xorot = 0xFFFFFFFF;
25:
26: cm_ini(p_cm);
27: for(i=0;i<len;i++)
28:     cm_nxt(p_cm,msg[i]);
29:
30: return cm_crc(p_cm);
31: }
32:
33: /*check msg to see if it is vaild*/
34: int valid(char* msg,int len)
35: {
36: int c_sum ;
37: int c_sum_given;
38: c_sum = checksum(msg,len-4);
39: memcpy (&c_sum_given,msg+len-4,4); /*grab checksum*/
```

```
40: /*printf("Expecting %x. Rec'd %x \n",c_sum_given,c_sum);*/  
41: if (c_sum==c_sum_given)  
42:     return TRUE; /*message is valid*/  
43: /*printf("Invalid checksum");*/  
44: return FALSE;  
45: }
```

File: sdm/common/checksum/crcmodel.h

```
1: /*****
2: /*          Start of crcmodel.h          */
3: /*****
4: /*
5: /* Author : Ross Williams (ross@guest.adelaide.edu.au.).    */
6: /* Date   : 3 June 1993.          */
7: /* Status : Public domain.        */
8: /*
9: /* Description : This is the header (.h) file for the reference */
10:
11: /* implementation of the Rocksoft^tm Model CRC Algorithm. For more */
12: /* information on the Rocksoft^tm Model CRC Algorithm, see the document */
13: /* titled "A Painless Guide to CRC Error Detection Algorithms" by Ross */
14: /* Williams (ross@guest.adelaide.edu.au.). This document is likely to be in */
15: /* "ftp.adelaide.edu.au/pub/rocksoft". */
16: /*
17: /* Note: Rocksoft is a trademark of Rocksoft Pty Ltd, Adelaide, Australia. */
18: /*
19: /*****/
20:
21: /*
22: /* How to Use This Package          */
23: /* -----          */
24: /* Step 1: Declare a variable of type cm_t. Declare another variable */
25: /* (p_cm say) of type p_cm_t and initialize it to point to the first*/
26: /* variable (e.g. p_cm_t p_cm = &cm_t).          */
27: /*
28: /* Step 2: Assign values to the parameter fields of the structure. */
29:
30: /* If you don't know what to assign, see the document cited earlier.*/
31: /* For example:          */
32: /*     p_cm->cm_width = 16;          */
33: /*     p_cm->cm_poly = 0x8005L;      */
34: /*     p_cm->cm_init = 0L;          */
35: /*     p_cm->cm_refin = TRUE;        */
36: /*     p_cm->cm_refot = TRUE;        */
37: /*     p_cm->cm_xorot = 0L;          */
38: /* Note: Poly is specified without its top bit (18005 becomes 8005).*/
39:
```



```

40: /*    Note: Width is one bit less than the raw poly width.    */
41: /*                                                                */
42: /* Step 3: Initialize the instance with a call cm_ini(p_cm);    */
43: /*                                                                */
44: /* Step 4: Process zero or more message bytes by placing zero or more    */
45: /*    successive calls to cm_nxt. Example: cm_nxt(p_cm,ch);    */
46: /*                                                                */
47: /* Step 5: Extract the CRC value at any time by calling crc = cm_crc(p_cm); */
48: /*    If the CRC is a 16-bit value, it will be in the bottom 16 bits. */
49:
50: /*                                                                */
51: /******
52: /*                                                                */
53: /* Design Notes    */
54: /* -----    */
55: /* PORTABILITY: This package has been coded very conservatively so that    */
56: /* it will run on as many machines as possible. For example, all external    */
57: /* identifiers have been restricted to 6 characters and all internal ones to    */
58: /* 8 characters. The prefix cm (for Crc Model) is used as an attempt to    */
59:
60: /* avoid namespace collisions. This package is endian independent.    */
61: /*                                                                */
62: /* EFFICIENCY: This package (and its interface) is not designed for    */
63: /* speed. The purpose of this package is to act as a well-defined reference    */
64: /* model for the specification of CRC algorithms. If you want speed, cook up    */
65: /* a specific table-driven implementation as described in the document cited    */
66: /* above. This package is designed for validation only; if you have found or    */
67: /* implemented a CRC algorithm and wish to describe it as a set of para-    */
68: /* meters to the Rocksofttm Model CRC Algorithm, your CRC algorithm imple-    */
69:
70: /* mentation should behave identically to this package under those para-    */
71: /* meters.    */
72: /*                                                                */
73: /******
74:
75: /* The following #ifndef encloses this entire    */
76: /* header file, rendering it idempotent.    */
77:
78: #ifndef CM_DONE
79:
80: #define CM_DONE

```

```

81:
82: #include "../sdmLib.h"
83:
84: /*****
85:  * The following definitions are extracted from my style header file which
86:  * would be cumbersome to distribute with this package. The DONE_STYLE is
87:  * the idempotence symbol used in my style header file.
88:
89: #ifndef DONE_STYLE
90:
91:
92: typedef unsigned long    ulong;
93: typedef unsigned        bool;
94: typedef unsigned char * p_ubyte_;
95:
96: #ifndef TRUE
97: #define FALSE 0
98: #define TRUE 1
99: #endif
100:
101:
102:  * Change to the second definition if you don't have prototypes.
103: #define P_(A) A
104:  * #define P_(A) ()
105:
106:  * Uncomment this definition if you don't have void.
107: typedef int void;
108:
109: #endif
110:
111:
112: /*****
113:  * CRC Model Abstract Type
114:  * -----
115:  * The following type stores the context of an executing instance of the
116:  * model algorithm. Most of the fields are model parameters which must be
117:  * set before the first initializing call to cm_ini.
118:
119: typedef struct
120: {
121:

```

```

122: int    cm_width; /* Parameter: Width in bits [8,32].    */
123: ulong cm_poly; /* Parameter: The algorithm's polynomial. */
124: ulong cm_init; /* Parameter: Initial register value.    */
125: bool  cm_refin; /* Parameter: Reflect input bytes?    */
126: bool  cm_refot; /* Parameter: Reflect output CRC?    */
127: ulong cm_xorot; /* Parameter: XOR this to output CRC. */
128:
129:
130: ulong cm_reg; /* Context: Context during execution. */
131: } cm_t;
132: typedef cm_t *p_cm_t;
133:
134: /*****
135:  /* Functions That Implement The Model
136:  /* -----
137:
138:  /* The following functions animate the cm_t abstraction.
139:
140:  extern void SDMLIB_API cm_ini P_((p_cm_t p_cm));
141:
142:  /* Initializes the argument CRC model instance.
143:  /* All parameter fields must be set before calling this.
144:
145:  extern void SDMLIB_API cm_nxt P_((p_cm_t p_cm,int ch));
146:
147:
148:  /* Processes a single message byte [0,255].
149:
150:  extern void SDMLIB_API cm_blk P_((p_cm_t p_cm,p_ubyte_ blk_adr,ulong blk_len));
151:
152:  /* Processes a block of message bytes.
153:
154:  extern ulong SDMLIB_API cm_crc P_((p_cm_t p_cm));
155:
156:  /* Returns the CRC value for the message bytes processed so far.
157:
158:
159:  *****/
160: /* Functions For Table Calculation
161:  /* -----
162:  /* The following function can be used to calculate a CRC lookup table.

```

```

163: /* It can also be used at run-time to create or check static tables. */
164:
165: extern ulong SDMLIB_API cm_tab P_((p_cm_t p_cm,int index));
166:
167:
168: /* Returns the i'th entry for the lookup table for the specified algorithm. */
169: /* The function examines the fields cm_width, cm_poly, cm_refin, and the */
170: /* argument table index in the range [0,255] and returns the table entry in */
171: /* the bottom cm_width bytes of the return value. */
172:
173: /*****
174: /* End of the header file idempotence #ifndef */
175:
176: #endif
177:
178:
179: /*****
180: /*          End of crcmodel.h */
181: /*****
182:

```

File: sdm/common/checksum/crcmodel.c

```
1: /*****
2: /*          Start of crcmodel.c          */
3: /*****
4: /*
5:
6: /* Author : Ross Williams (ross@guest.adelaide.edu.au.).    */
7: /* Date   : 3 June 1993.                                     */
8: /* Status : Public domain.                                   */
9: /*
10: /* Description : This is the implementation (.c) file for the reference */
11: /* implementation of the Rocksoft^tm Model CRC Algorithm. For more    */
12: /* information on the Rocksoft^tm Model CRC Algorithm, see the document */
13: /* titled "A Painless Guide to CRC Error Detection Algorithms" by Ross */
14: /* Williams (ross@guest.adelaide.edu.au.). This document is likely to be in */
15:
16: /* "ftp.adelaide.edu.au/pub/rocksoft".    */
17: /*
18: /* Note: Rocksoft is a trademark of Rocksoft Pty Ltd, Adelaide, Australia. */
19: /*
20: /*****
21: /*
22: /* Implementation Notes          */
23: /* -----          */
24: /* To avoid inconsistencies, the specification of each function is not */
25:
26: /* echoed here. See the header file for a description of these functions. */
27: /* This package is light on checking because I want to keep it short and */
28: /* simple and portable (i.e. it would be too messy to distribute my entire */
29: /* C culture (e.g. assertions package) with this package.          */
30: /*
31: /*****
32:
33: #include "crcmodel.h"
34:
35:
36: /*****
37: /* The following definitions make the code more readable.          */
38:
39: #define BITMASK(X) (1L << (X))
```

```

40: #define MASK32 0xFFFFFFFFL
41: #define LOCAL static
42:
43: /*****
44:
45:
46: LOCAL ulong reflect P_((ulong v,int b));
47: LOCAL ulong reflect (v,b)
48:
49: /* Returns the value v with the bottom b [0,32] bits reflected. */
50: /* Example: reflect(0x3e23L,3) == 0x3e26          */
51:
52: ulong v;
53: int  b;
54:
55: {
56: int  i;
57: ulong t = v;
58: for (i=0; i<b; i++)
59: {
60: if (t & 1L)
61: v|= BITMASK((b-1)-i);
62: else
63: v&= ~BITMASK((b-1)-i);
64: t>>=1;
65: }
66: return v;
67: }
68:
69: /*****
70:
71: LOCAL ulong widmask P_((p_cm_t));
72: LOCAL ulong widmask (p_cm)
73:
74: /* Returns a longword whose value is (2^p_cm->cm_width)-1. */
75: /* The trick is to do this portably (e.g. without doing <<32). */
76: p_cm_t p_cm;
77: {
78: return (((1L<<(p_cm->cm_width-1))-1L)<<1)|1L;
79:
80: }

```

```

81:
82: /*****
83:
84: void cm_ini (p_cm)
85: p_cm_t p_cm;
86: {
87: p_cm->cm_reg = p_cm->cm_init;
88:
89: }
90:
91: /*****
92:
93: void cm_nxt (p_cm,ch)
94: p_cm_t p_cm;
95: int ch;
96: {
97: int i;
98:
99: ulong uch = (ulong) ch;
100: ulong topbit = BITMASK(p_cm->cm_width-1);
101:
102: if (p_cm->cm_refin) uch = reflect(uch,8);
103: p_cm->cm_reg ^= (uch << (p_cm->cm_width-8));
104: for (i=0; i<8; i++)
105:
106: {
107: if (p_cm->cm_reg & topbit)
108: p_cm->cm_reg = (p_cm->cm_reg << 1) ^ p_cm->cm_poly;
109: else
110: p_cm->cm_reg <<= 1;
111:
112: p_cm->cm_reg &= widmask(p_cm);
113: }
114: }
115:
116: /*****
117:
118: void cm_blk (p_cm,blk_adr,blk_len)
119:
120: p_cm_t p_cm;
121: p_ubyte blk_adr;

```

```

122: ulong blk_len;
123: {
124: while (blk_len--) cm_nxt(p_cm,*blk_adr++);
125: }
126:
127: /*****
128:
129:
130: ulong cm_crc (p_cm)
131: p_cm_t p_cm;
132: {
133: if (p_cm->cm_refot)
134: return p_cm->cm_xorot ^ reflect(p_cm->cm_reg,p_cm->cm_width);
135: else
136:
137: return p_cm->cm_xorot ^ p_cm->cm_reg;
138: }
139:
140: /*****
141:
142: ulong cm_tab (p_cm,index)
143: p_cm_t p_cm;
144: int index;
145:
146: {
147: int i;
148: ulong r;
149: ulong topbit = BITMASK(p_cm->cm_width-1);
150: ulong inbyte = (ulong) index;
151:
152: if (p_cm->cm_refin) inbyte = reflect(inbyte,8);
153:
154: r = inbyte << (p_cm->cm_width-8);
155: for (i=0; i<8; i++)
156: if (r & topbit)
157: r = (r << 1) ^ p_cm->cm_poly;
158:
159: else
160: r<<=1;
161: if (p_cm->cm_refin) r = reflect(r,p_cm->cm_width);
162: return r & widmask(p_cm);

```



```
163: }
164:
165:
166: /*****
167:      End of crcmodel.c
168: *****/
169:
```

File: sdm/common/checksum/Makefile

```
1: # Makefile for CRC check system
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: .PHONY:    all clean distclean
7:
8: all: checksum.o crcmodel.o
9:
10: checksum.o:    checksum.c
11: $(CC) $(CFLAGS) -fPIC -c $<
12:
13: crcmodel.o: crcmodel.c
14: $(CC) $(CFLAGS) -fPIC -c $<
15:
16: clean:
17: rm -f *.o *~
18:
19: distclean: clean
```

File: sdm/common/checksum/checksum.h

```
1: //header for functions to do checksums for SDM messages
2:
3: #include "../sdmLib.h"
4:
5: #ifdef __cplusplus
6: extern "C" {
7: #endif
8:
9: extern SDMLIB_API int checksum(char* msg,int len);
10: extern SDMLIB_API int checksum_valid(char* msg,int len);
11:
12: #ifdef __cplusplus
13: }
14: #endif
```

File: sdm/common/SubscriptionManager/SubscriptionManager.h

```
1: //a class to handle subscriptions and data publishing
2:
3: #ifndef __SUBSCRIPTION_MANAGER_H_
4: #define __SUBSCRIPTION_MANAGER_H_
5:
6: #include "../message/SDMmessage.h"
7: #include "../message/SDMSubreqst.h"
8: #include "../message/SDMSerreqst.h"
9: #include "../message/SDMDeletesub.h"
10: #include "../message/SDMData.h"
11: #include "../message/SDMComponent_ID.h"
12: #include "../message/SDMMessage_ID.h"
13:
14: #include "../sdmLib.h"
15:
16: typedef struct sub
17: {
18:     SDMComponent_ID component_id;
19:     SDMMMessage_ID msg_id;
20:     SDMMMessage_ID fault_id;
21:     bool isSerreqst;
22:     struct sub* next;
23:     sub():component_id(),msg_id(),fault_id(),isSerreqst(false),next(0) {}
24:     sub(const sub& r);
25:     sub& operator=(const sub& r);
26: } subscription;
27:
28:
29: class SDMLIB_API SubscriptionManager
30: {
31: public:
32:     SubscriptionManager();
33:     SubscriptionManager(const SubscriptionManager&);
34:     ~SubscriptionManager();
35:     bool AddSubscription(SDMSubreqst &subreqst);
36:     bool AddSubscription(SDMSerreqst &serreqst);
37:     bool RemoveSubscription(SDMDeletesub &deletesub);
38:     int SubscriptionCount(void);
39:     bool ClearAllSubscriptions(void);
```

```
40: bool Publish(unsigned char interface_id,unsigned char message_id,char* data,long length);
41: bool Publish(SDMMMessage_ID id, char* data, long length);
42: SubscriptionManager& operator=(const SubscriptionManager&);
43: SDMData* GetLastPublished(void);
44: private:
45: SDMComponent_ID my_component_id;
46: subscription* sub_list;
47: SDMData dat;
48: };
49:
50: #endif
```

File: sdm/common/SubscriptionManager/SubscriptionManager.cpp

```
1: #include "SubscriptionManager.h"
2: #include "../message/SDMData.h"
3: #include <string.h>
4: #include <stdio.h>
5:
6: void delete_subptr(subscription*);
7: subscription* deep_subcopy(subscription*);
8:
9: SubscriptionManager::SubscriptionManager():my_component_id(),sub_list(NULL),dat()
10: {};
11:
12: SubscriptionManager::SubscriptionManager(const SubscriptionManager&
a):my_component_id(a.my_component_id),sub_list(NULL),dat()
13: {
14: sub_list = deep_subcopy(a.sub_list);
15: }
16:
17: SubscriptionManager::~SubscriptionManager()
18: {
19: delete_subptr(sub_list);
20: }
21:
22: bool SubscriptionManager::AddSubscription(SDMSubreqst &subreqst)
23: {
24: subscription* new_sub = NULL;
25: //look for subscription
26: for(subscription* cur=sub_list;cur!=NULL;cur=cur->next)
27: {
28:     if(cur->msg_id == subreqst.msg_id)
29:         if(cur->component_id.getAddress() == subreqst.destination.getAddress())
30:             if(cur->component_id.getPort() == subreqst.destination.getPort())
31:                 {
32:                     return false; //duplicate subscription
33:                 }
34: }
35: new_sub = new(subscription);
36: new_sub->msg_id = subreqst.msg_id;
37: new_sub->fault_id = subreqst.fault_id;
38: new_sub->component_id = subreqst.destination;
```

```

39: new_sub->isSerreqst = false;
40: new_sub->next = sub_list;
41: sub_list = new_sub;
42: my_component_id = subreqst.source;
43: return true;
44: }
45:
46: bool SubscriptionManager::AddSubscription(SDMSerreqst &serreqst)
47: {
48: subscription* new_sub = NULL;
49: //look for subscription
50: for(subscription* cur=sub_list;cur!=NULL;cur=cur->next)
51: {
52:     if(cur->msg_id == serreqst.reply_id)
53:         if(cur->component_id.getAddress() == serreqst.destination.getAddress())
54:             if(cur->component_id.getPort() == serreqst.destination.getPort())
55:                 {
56:                     return false; //duplicate subscription
57:                 }
58: }
59: new_sub = new(subscription);
60: new_sub->msg_id = serreqst.reply_id;
61: new_sub->fault_id = serreqst.fault_id;
62: new_sub->component_id = serreqst.destination;
63: new_sub->isSerreqst = true;
64: new_sub->next = sub_list;
65: sub_list = new_sub;
66: my_component_id = serreqst.source;
67: return true;
68: }
69:
70: bool SubscriptionManager::RemoveSubscription(SDMDelatesub &delatesub)
71: {
72: subscription* prev = NULL;
73: //look for subscription
74: for(subscription* cur=sub_list;cur!=NULL;cur=cur->next)
75: {
76:     if(cur->msg_id == delatesub.msg_id)
77:         if(cur->component_id.getAddress() == delatesub.destination.getAddress())
78:             if(cur->component_id.getPort() == delatesub.destination.getPort())
79:                 {

```

```

80:         if(prev == NULL)
81:             sub_list = cur->next;
82:         else
83:             prev->next = cur->next;
84:         delete cur;
85:         return true;
86:     }
87:     prev = cur;
88: }
89: return false;
90: }
91:
92:
93: int SubscriptionManager::SubscriptionCount(void)
94: {
95:     int count = 0;
96:     for(subscription* cur=sub_list;cur!=NULL;cur=cur->next)
97:     {
98:         count++;
99:     }
100:     return count;
101: }
102:
103: bool SubscriptionManager::Publish(unsigned char interface_id,unsigned char message_id,char*
data,long length)
104: {
105:     SDMMMessage_ID id(interface_id, message_id);
106:     return Publish (id, data, length);
107:
108: }
109:
110: bool SubscriptionManager::Publish(SDMMMessage_ID id, char* data, long length)
111: {
112:     bool published = false;
113:     int result;
114:     //form data message
115:     dat.source = my_component_id;
116:     dat.msg_id = id;
117:     memcpy(dat.msg,data,length);
118:     subscription *prev = NULL;
119:     //search for matching subscriptions

```



```

120:   for(subscription* cur=sub_list;cur!=NULL;prev=cur, cur=cur->next)
121:   {
122:       if((cur->msg_id == id) || (cur->fault_id == id))
123:       {
124:           result = dat.Send(cur->component_id,length);
125:           if(result != -1)
126:           {
127:               //If this is a service request (i.e. one-time message response), remove the item upon
first publish
128:               if (cur->isSerreqst)
129:               {
130:                   //if cur is the list head
131:                   if (prev == NULL)
132:                   {
133:                       sub_list = cur->next;
134:                       delete cur;
135:                   }
136:                   //if cur is in the middle or at the end of the list
137:                   else
138:                   {
139:                       prev->next = cur->next;
140:                       delete cur;
141:                   }
142:               }
143:               published = true;
144:           }
145:       }
146:   }
147:   return published;
148: }
149:
150: SubscriptionManager& SubscriptionManager::operator=(const SubscriptionManager& a)
151: {
152:     my_component_id = a.my_component_id;
153:     sub_list = deep_subcopy(a.sub_list);
154:     return *this;
155: }
156:
157: bool SubscriptionManager::ClearAllSubscriptions(void)
158: {
159:     if (sub_list == NULL) return false;

```

```

160:    delete_subptr(sub_list);
161:    sub_list = NULL;
162:    return true;
163: }
164:
165: SDMDData *SubscriptionManager::GetLastPublished()
166: {
167:     return &dat;
168: }
169:
170: void delete_subptr(subscription *p)
171: {
172:     if (p == NULL) return;
173:     delete_subptr(p->next);
174:     delete(p);
175: }
176:
177: subscription* deep_subcopy(subscription *p)
178: {
179:     subscription* q = NULL;
180:     if (p == NULL) return NULL;
181:     q = new(subscription);
182:     q->msg_id = p->msg_id;
183:     q->fault_id = q->fault_id;
184:     //q->ip_addr = p->ip_addr;
185:     //q->port = p->port;
186:     q->isSerreqst = p->isSerreqst;
187:     q->component_id = p->component_id;
188:     q->next = deep_subcopy(p->next);
189:     return q;
190: }
191:

```

File: sdm/common/SubscriptionManager/Makefile

```
1: #SubscriptionManager makefile
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: .PHONY:    all clean distclean
7:
8: all:    SubscriptionManager.o
9:
10: SubscriptionManager.o: SubscriptionManager.cpp SubscriptionManager.h
11: $(CXX) $(CXXFLAGS) -fPIC -c $<
12:
13: clean:
14: rm -f *.o *~
15:
16: distclean:    clean
```

File: sdm/common/MessageManager/Makefile

```
1: #MessageManager makefile
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: .PHONY:    all clean distclean
7:
8: all:    MessageManager.o
9:
10: MessageManager.o: MessageManager.cpp MessageManager.h
11: $(CXX) $(CXXFLAGS) -fPIC -c $<
12:
13: clean:
14: rm -f *.o *~
15:
16: distclean:    clean
```

File: sdm/common/MessageManager/MessageManager.cpp

```
1: #include "MessageManager.h"
2:
3: #include "../message/SDMmessage.h"
4: #include "../message/SDMCommand.h"
5:
6: #include "../UDPcom.h"
7: #include "../TCPcom.h"
8: #include "../ErrorUtils.h"
9:
10: #include <pthread.h>
11: #include <string.h>
12: #include <unistd.h>
13: #include <sys/types.h>
14: #include <sys/socket.h>
15: #include <netinet/in.h>
16: #include <arpa/inet.h>
17: #include <stdio.h>
18: #include <signal.h>
19:
20: // #define DEBUG_MESSAGE_MANAGER 1
21: #define MM_LISTEN_STACKSZ (65536)
22: pthread_mutex_t message_manager_lock = PTHREAD_MUTEX_INITIALIZER;
23:
24: MessageManager::MessageManager(): m_port(-
1), handlerThread(), tcpThread(), forward_component_id(), forward_command_id(0,0), forward_callback(N
ULL), UDPsock(0), TCPsock(0), queued(0), max_queued(0), UsingTCP(false), UsingUDP(false)/*, both(fals
e)*/, init_success(false), queue_head(NULL), queue_tail(NULL), semMsgWaitQ(0), semStartup(0),
semStartupTCP(0)
25: {
26: }
27:
28: // NOTE: the copy constructor preforms a _shallow_ copy of the data, so that if a Listen thread has
been begun any messages received will be available in the copy.
29: MessageManager::MessageManager(const MessageManager&
a): m_port(a.m_port), handlerThread(a.handlerThread), tcpThread(a.tcpThread), forward_component_id(a.f
orward_component_id), forward_command_id(a.forward_command_id), forward_callback(a.forward_call
back), UDPsock(a.UDPsock), TCPsock(a.TCPsock), queued(a.queued), max_queued(a.max_queued), Using
TCP(a.UsingTCP), UsingUDP(a.UsingUDP)/*, both(a.both)*/, init_success(a.init_success), queue_head(a.q
ueue_head), queue_tail(a.queue_tail), semMsgWaitQ(a.semMsgWaitQ), semStartup(a.semStartup),
semStartupTCP(a.semStartupTCP)
30: {}
```

```

31:
32: MessageManager::~MessageManager()
33: {
34: #ifdef DEBUG_MESSAGESMANAGER
35:     printf("MessageManager::~MessageManager \n");
36: #endif
37:
38: //Cancel the listener threads
39: if (UsingTCP)
40: {
41:     pthread_cancel(tcpThread);
42:     TCPshutdown(TCPsock); // stop blocking call so the thread can detect cancellation
43:     TCPclose(TCPsock);
44:     pthread_join(handlerThread, NULL);
45: }
46: if(UsingUDP)
47: {
48:     pthread_cancel(handlerThread);
49:     UDPshutdown(UDPsock); // stop blocking call so the thread can detect cancellation
50:     UDPclose(UDPsock);
51:     pthread_join(handlerThread, NULL);
52: }
53: }
54:
55: void* MessageManager::Listen(void* arg)
56: {
57: MessageManager* MM_ptr = (MessageManager*)arg;
58: char buf[BUFSIZE];
59: long length = -1;
60: #ifndef WIN32
61: sigset_t signal_set;
62: #endif
63: struct sockaddr_in s;
64:
65: MM_ptr->init_success = true;
66: #ifdef DEBUG_MESSAGESMANAGER
67: printf("UDP Message Manager binding port %ld \n",MM_ptr->m_port);
68: #endif
69: #ifndef WIN32
70: sigfillset(&signal_set);
71: pthread_sigmask(SIG_BLOCK, &signal_set, NULL);

```

```

72: #endif
73: MM_ptr->UDPSock = UDPpassive(MM_ptr->m_port);
74: if(MM_ptr->UDPSock == IP_SOCK_INVALID)
75:     MM_ptr->init_success = false;
76:
77: MM_ptr->semStartup.Signal();
78: if(MM_ptr->init_success == false)
79: {
80: #ifdef DEBUG_MESSAGE_MANAGER
81:     printf("MM: %s -- Error binding port. \n", __FUNCTION__);
82: #endif
83:     return NULL;
84: }
85:
86: while(1)
87: {
88:     length = UDPServ_recv(MM_ptr->UDPSock, buf, sizeof(buf));
89:     pthread_testcancel();
90:     if (length < 0)
91:         continue;
92: #ifdef DEBUG_MESSAGE_MANAGER
93:     printf("UDP Message Manger rec'd message 0x%hhx \n", buf[0]);
94: #endif
95:     UDPgetip(&s);
96:     MM_ptr->AddMessage(buf,length,MM_ptr->UDPSock,false,&s);
97: //     memset(buf,0,sizeof(buf));
98: }
99: return NULL;
100: }
101:
102: void* MessageManager::ListenTCP(void* arg)
103: {
104:     MessageManager* MM_ptr = (MessageManager*)arg;
105:     char buf[LARGE_MSG_BUFSIZE];
106:     short sSdmLength;
107:     struct sockaddr_in sin;
108:     int iStatus, iCurLength;
109:     int tcpSock = 0;
110:     bool bError = false;
111: #ifndef WIN32
112:     sigset_t signal_set;

```

```

113: #endif
114:
115:     MM_ptr->init_success = true;
116: #ifdef DEBUG_MESSAGE_MANAGER
117:     printf("TCP Message Manager binding port %ld \n",MM_ptr->m_port);
118: #endif
119: #ifndef WIN32
120:     sigfillset(&signal_set);
121:     pthread_sigmask(SIG_BLOCK, &signal_set, NULL);
122: #endif
123:     MM_ptr->TCPsock = TCPpassive(MM_ptr->m_port, MAX_TCP_CONNECTIONS);
124:     if (MM_ptr->TCPsock == IP_SOCKET_INVALID)
125:         MM_ptr->init_success = false;
126:
127:     MM_ptr->semStartupTCP.Signal();
128:     if(MM_ptr->init_success == false)
129:         return NULL;
130:
131:     while(1)
132:     {
133:         tcpSock = TCPaccept(MM_ptr->TCPsock, &sin);
134:         if (tcpSock == IP_SOCKET_INVALID)
135:             continue;
136:         pthread_testcancel();
137:         //
138:         // Receive the header of the message
139:         bError = false;
140:         iCurLength = 0;
141:         do
142:         {
143:             iStatus = TCPrecv(tcpSock, &buf[iCurLength], LARGE_MSG_BUFSIZE - iCurLength);
144:             if (iStatus < 0)
145:             {
146:                 perror("TCPrecv");
147:                 bError = true;
148:             }
149:             else if (iStatus == 0)
150:                 bError = true;
151:             else
152:                 iCurLength += iStatus;
153:         } while(iCurLength < HEADER_SIZE);

```



```

154:     if (bError)
155:     {
156:         TCPclose(tcpSock);
157:         continue;
158:     }
159:     sSdmLength = GET_SHORT(&buf[9]);
160:     //
161:     // Receive the rest of the SDM message
162:     while(sSdmLength > iCurLength - HEADER_SIZE)
163:     {
164:         iStatus = TCPrecv(tcpSock, &buf[iCurLength], LARGE_MSG_BUFSIZE - iCurLength);
165:         if (iStatus < 0)
166:         {
167:             perror("TCPrecv");
168:             bError = true;
169:         }
170:         else if (iStatus == 0)
171:             bError = true;
172:         else
173:             iCurLength += iStatus;
174:     }
175:     if (bError)
176:     {
177:         TCPclose(tcpSock);
178:         continue;
179:     }
180:
181:     if(sSdmLength != iCurLength - HEADER_SIZE)
182:         printf("Error: More bytes recv'd than expected! \n");
183: #ifdef DEBUG_MESSAGE_MANAGER
184:     printf("TCP Message Manger rec'd message \n");
185: #endif
186:     MM_ptr->AddMessage(buf,iCurLength,tcpSock,true,&sin);
187: //     memset(buf,0,sizeof(buf));
188: }
189: return NULL;
190: }
191:
192:
193:
194:

```

195: //NOTE: the = operator preforms a _shallow_ copy of the data, so that if a Listen thread has been begun any messages received will be available in the copy.

196: MessageManager& MessageManager::operator=(const MessageManager& a)

197: {

198: handlerThread = a.handlerThread;

199: m_port = a.m_port;

200: forward_component_id=a.forward_component_id;

201: forward_command_id=a.forward_command_id;

202: forward_callback=a.forward_callback;

203: UDPsock=a.UDPsock;TCPsock=a.TCPsock;queued=a.queued;

204: max_queued=a.max_queued;

205: queue_head=a.queue_head;

206: queue_tail=a.queue_tail;

207: semMsgWaitQ=a.semMsgWaitQ;

208: semStartup=a.semStartup;

209: return *this;

210: }

211: /*

212: * Initialize the UDP listener thread for the MessageManager.

213: */

214: bool MessageManager::Async_Init(long port)

215: {

216: pthread_attr_t attr;

217: pthread_attr_init(&attr);

218: pthread_attr_setstacksize(&attr,MM_LISTEN_STACKSZ);

219: pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);

220:

221: //If we are already initialized, return

222: if(m_port != -1 && (!UsingTCP && !UsingUDP))

223: {

224: return false;

225: }

226:

227: m_port = port;

228: if (pthread_create(&handlerThread,&attr,&Listen,this) < 0)

229: {

230: printf("MessageManager::Error creating UDP listener thread. \n");

231: return false;

232: }

233: UsingUDP = true;

234: semStartup.Wait();

```

235:
236:     return init_success;
237: }
238: /*
239:  * Initialize the TCP listener thread for the MessageManager.
240:  */
241: bool MessageManager::Async_Init_TCP(long port)
242: {
243:     pthread_attr_t attr;
244:     pthread_attr_init(&attr);
245:     pthread_attr_setstacksize(&attr,MM_LISTEN_STACKSZ);
246:     pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
247:
248:     //If we are already initialized, return
249:     if(m_port != -1 && (!UsingTCP && !UsingUDP))
250:     {
251:         return false;
252:     }
253:
254:     m_port = port;
255:     if(pthread_create(&tcpThread,&attr,&ListenTCP,this) < 0)
256:     {
257:         printf("MessageManager::Error creating TCP listener thread. \n");
258:         return false;
259:     }
260:     UsingTCP = true;
261:     semStartupTCP.Wait();
262:     return init_success;
263: }
264: /*
265:  * Initialize both the TCP and UDP listener threads for the MessageManager.
266:  */
267: bool MessageManager::Async_Init_Both(long port)
268: {
269:     if(Async_Init(port)==false)
270:         return false;
271:     if(Async_Init_TCP(port)==false)
272:         return false;
273:     UsingTCP = UsingUDP = true;
274:     return true;
275: }

```

```

276:
277:     void      MessageManager::SetForwarding(long      count,SDMComponent_ID
component_id,SDMMessage_ID command_id,short (*callback) (char*,long))
278: {
279:     pthread_mutex_lock(&message_manager_lock);
280:
281:     max_queued = count;
282:     forward_component_id = component_id;
283:     forward_command_id = command_id;
284:     forward_callback = callback;
285:
286:     pthread_mutex_unlock(&message_manager_lock);
287: }
288:
289: bool MessageManager::IsReady()
290: {
291:     bool ready = false;
292:
293:     pthread_mutex_lock(&message_manager_lock);
294:
295:     if(queued > 0)
296:     {
297:         ready = true;
298:     }
299:
300:     pthread_mutex_unlock(&message_manager_lock);
301:     return ready;
302: }
303:
304: message_item* MessageManager::GetHeadEntry()
305: {
306:     message_item *temp = NULL;
307:
308:     pthread_mutex_lock(&message_manager_lock);
309:     if (queue_head != NULL)
310:     {
311:         temp = queue_head;
312:         queue_head = queue_head->next;
313:
314:         if (queue_head == NULL)
315:             queue_tail = NULL;

```

```

316:     --queued;
317: }
318: else
319:     printf("MessageManager::GetMsgEntry: Unexpected NULL \n");
320:
321:     pthread_mutex_unlock(&message_manager_lock);
322:     return temp;
323: }
324:
325: #ifdef WIN32
326: char MessageManager::GetMsg(char *buf, long& length)
327: #else
328: char MessageManager::GetMessage(char* buf,long& length)
329: #endif
330: {
331:     char msg_type = 0;
332:     message_item* temp;
333:     length = 0;
334:
335:     if( semMsgWaitQ.TryWait() == 0 )
336:     {
337:         temp = GetHeadEntry();
338:         if (temp != NULL)
339:         {
340:             memcpy(buf,temp->msg,temp->length);
341:             length = temp->length;
342:             msg_type = buf[0];
343:
344:             delete[] temp->msg;
345:             delete temp->sin;
346:             delete temp;
347:         }
348:     }
349:     return msg_type;
350: }
351: #ifdef WIN32
352: char MessageManager::GetMsg(char * buf)
353: #else
354: char MessageManager::GetMessage(char* buf)
355: #endif
356: {

```

```

357: long iLength;
358: #ifdef WIN32
359: return GetMsg(buf, iLength);
360: #else
361: return GetMessage(buf, iLength);
362: #endif
363: }
364: #ifdef WIN32
365: char MessageManager::GetMsg(char* buf,long& length, SDMComHandle& HandleOut)
366: #else
367: char MessageManager::GetMessage(char* buf,long& length, SDMComHandle& HandleOut)
368: #endif
369: {
370:     char msg_type = 0;
371:     message_item* temp;
372:     length = 0;
373:
374:     if( semMsgWaitQ.TryWait() == 0 )
375:     {
376:         temp = GetHeadEntry();
377:         if (temp != NULL)
378:         {
379:             memcpy(buf,temp->msg,temp->length);
380:             length = temp->length;
381:             msg_type = buf[0];
382:
383:             HandleOut.Set(temp->socket, temp->tcp, temp->sin);
384:
385:             delete[] temp->msg;
386:             delete temp->sin;
387:             delete temp;
388:         }
389:     }
390:     return msg_type;
391: }
392:
393: char MessageManager::BlockGetMessage(char* buf,long& length)
394: {
395:     char msg_type = 0;
396:     message_item* temp;
397:     length = 0;

```

```

398:
399:  if( 0 == semMsgWaitQ.Wait() )
400:  {
401:      temp = GetHeadEntry();
402:      if (temp != NULL)
403:      {
404:          memcpy(buf,temp->msg,temp->length);
405:          length = temp->length;
406:          delete[] temp->msg;
407:          delete temp->sin;
408:          delete temp;
409:          msg_type = buf[0];
410:      }
411:  }
412:  else
413:  {
414:      perror("MessageManager::BlockGetMessage::Sem::Wait");
415:  }
416:
417:  return msg_type;
418: }
419:
420: char MessageManager::BlockGetMessage(char* buf, long& length, SDMComHandle& HandleOut)
421: {
422:     char msg_type = 0;
423:     message_item* temp;
424:     length = 0;
425:
426:     if( semMsgWaitQ.Wait() == 0 )
427:     {
428:         temp = GetHeadEntry();
429:
430:         memcpy(buf,temp->msg,temp->length);
431:         length = temp->length;
432:         msg_type = buf[0];
433:
434:         HandleOut.Set(temp->socket, temp->tcp, temp->sin);
435:
436:         delete[] temp->msg;
437:         delete temp->sin;
438:         delete temp;

```

```

439:     }
440:     return msg_type;
441: }
442:
443: char MessageManager::BlockGetMessage(char* buf)
444: {
445:     long iLength;
446:
447:     return BlockGetMessage(buf, iLength);
448: }
449:
450:
451:
452: void MessageManager::AddMessage(char* buf, long length, int socket, bool tcp, struct sockaddr_in*
sin)
453: {
454:     pthread_mutex_lock(&message_manager_lock);
455:     if ((max_queued != 0)&&(queued >= max_queued))
456:     {
457:         //forward message
458:         SDMCommand forward;
459:         forward.length = (length > (long)sizeof(forward.data) ? (long)sizeof(forward.data) : length);
460:         if (forward_callback!= NULL)
461:             forward.length = forward_callback(buf,length);
462:         forward.source = forward_component_id;
463:         forward.command_id = forward_command_id;
464:         memcpy(forward.data,buf,(forward.length > (long)sizeof(forward.data) ?
(long)sizeof(forward.data) : forward.length));
465:         forward.Send();
466:     }
467:     else
468:     {
469:         //queue message
470:         message_item* new_msg = new message_item;
471:         if (new_msg == NULL)
472:             ErrorUtils::MemoryAllocError(__FUNCTION__);
473:
474:         new_msg->msg = new char[length];
475:         if (new_msg->msg == NULL)
476:             ErrorUtils::MemoryAllocError(__FUNCTION__);
477:

```



```

478:     memcpy(new_msg->msg,buf,length);
479:     new_msg->length = length;
480:     new_msg->socket = socket;
481:     new_msg->tcp = tcp;
482:     new_msg->sin = new sockaddr_in;
483:     if (new_msg->sin == NULL)
484:         ErrorUtils::MemoryAllocError(__FUNCTION__);
485:
486:     new_msg->sin->sin_family = sin->sin_family;
487:     new_msg->sin->sin_port = sin->sin_port;
488:     new_msg->sin->sin_addr = sin->sin_addr;
489:     new_msg->next = NULL;
490:     if (NULL == queue_tail)
491:     {
492:         queue_tail = new_msg;
493:         queue_head = queue_tail;
494:     }
495:     else
496:     {
497:         queue_tail->next = new_msg;
498:         queue_tail = new_msg;
499:     }
500:     semMsgWaitQ.Signal();
501:     ++queued;
502: }
503: pthread_mutex_unlock(&message_manager_lock);
504: }

```

File: sdm/common/MessageManager/MessageManager.h

```
1: #ifndef __MESSAGE_MANAGER_H_
2: #define __MESSAGE_MANAGER_H_
3:
4: #include "../message/SDMmessage.h"
5: #include "../message/SDMComponent_ID.h"
6: #include "../message/SDMMessage_ID.h"
7: #include "../semaphore/semaphore.h"
8: #include "../SDMComHandle.h"
9: #include "../sdmLib.h"
10:
11: typedef struct msg_list_node
12: {
13: char* msg;
14: int length;
15: int socket;
16: bool tcp;
17: struct sockaddr_in *sin;
18: struct msg_list_node* next;
19: } message_item;
20:
21: class SDMLIB_API MessageManager
22: {
23: public:
24: MessageManager();
25: MessageManager(const MessageManager&);
26: ~MessageManager();
27:
28: MessageManager& operator=(const MessageManager&);
29:
30: bool Async_Init(long port);
31: bool Async_Init_TCP(long port);
32: bool Async_Init_Both(long port);
33: bool IsReady();
34: /*Windows has a WIN32 API function called GetMessage, so this needs to be renamed under
Windows */
35: #ifdef WIN32
36: char GetMsg(char * buf);
37: char GetMsg(char * buf, long& length);
38: char GetMsg(char* buf,long& length, SDMComHandle& HandleOut /* output param */);
```

```

39: #else
40: char GetMessage(char* buf);
41: char GetMessage(char* buf,long& length);
42: char GetMessage(char* buf,long& length, SDMComHandle& HandleOut /* output param */);
43: #endif
44: char BlockGetMessage(char* buf);
45: char BlockGetMessage(char* buf,long& length);
46: char BlockGetMessage(char* buf, long& length, SDMComHandle& HandleOut /* output param */);
47: void AddMessage(char* buf,long length,int socket,bool tcp,struct sockaddr_in* sin);
48: void      SetForwarding(long      count,SDMComponent_ID      component_id,SDMMMessage_ID
command_id,short (*callback) (char*,long));
49: long m_port;
50: private:
51: static void* Listen(void* arg);
52: static void* ListenTCP(void* arg);
53:
54: pthread_t handlerThread, tcpThread;
55: SDMComponent_ID forward_component_id;
56: SDMMMessage_ID forward_command_id;
57: short (*forward_callback) (char*,long);
58: int UDPsock;
59: int TCPsock;
60: int queued;
61: int max_queued;
62: bool UsingTCP;           //If true, the MM is listening on TCP
63: bool UsingUDP;           //If true, the MM is listening on UDP
64: bool init_success;
65: message_item* queue_head;
66: message_item* queue_tail;
67: message_item* GetHeadEntry();
68:
69: Sem semMsgWaitQ;
70: Sem semStartup;
71: Sem semStartupTCP;
72: };
73:
74: #endif

```

File: sdm/common/task/SDMTaskResources.cpp

```
1: #include "SDMTaskResources.h"
2:
3: SDMTaskResources::SDMTaskResources() : m_usResources(0)
4: {
5: }
6:
7: SDMTaskResources::SDMTaskResources(unsigned short usResources) : m_usResources(usResources)
8: {
9: }
10:
11: SDMTaskResources::SDMTaskResources(const SDMTaskResources& other) :
m_usResources(other.m_usResources)
12: {
13: }
14:
15: SDMTaskResources& SDMTaskResources::operator=(const SDMTaskResources& other)
16: {
17: m_usResources = other.m_usResources;
18: return *this;
19: }
20:
21: void SDMTaskResources::Set(unsigned short usResources)
22: {
23: m_usResources = usResources;
24: }
25:
26: bool SDMTaskResources::MatchResources (const SDMTaskResources& pmResources, const
SDMTaskResources& taskResources)
27: {
28: if (taskResources.IsPreferredPmIdSet())
29:     if (pmResources.GetPreferredPmId() != taskResources.GetPreferredPmId())
30:         return false;
31:
32: if (pmResources.GetArch() == taskResources.GetArch())
33:     if (pmResources.GetMem() >= taskResources.GetMem())
34:         if (pmResources.GetOs() == taskResources.GetOs())
35:             return true;
36: return false;
37: }
```

File: sdm/common/task/Makefile

```
1:
2: include ../../Makefile.common
3: include ../../$(MAKEFILE_DEFS)
4:
5: .PHONY: all clean distclean
6:
7: all:   SDMTaskResources.o
8:
9: SDMTaskResources.o:  SDMTaskResources.cpp SDMTaskResources.h
10: $(CXX) $(CXXFLAGS) -fPIC -c $<
11:
12: clean:
13: rm -f *.o *~
14:
15: distclean:  clean
```

File: sdm/common/task/tasklist.h

```
1: // TaskList definition file
2: #ifndef __TASKLIST_H_
3: #define __TASKLIST_H_
4:
5: #include "task.h"
6: #include "../sdmLib.h"
7:
8: class SDMLIB_API TaskList
9: {
10: public:
11: TaskList();
12: void Init();
13: bool AnyInactive();
14: bool AnyPending();
15: bool AddTo(const Task& t);
16: bool RemoveFrom(unsigned int pid);
17: bool RemoveFrom(char *filename);
18: bool TaskFinished(unsigned int a_uiPid);
19: bool IsFilePresent(char *filename);
20: bool SetAddress(unsigned int pid, const SDMComponent_ID& PM);
21: unsigned int SetState (unsigned int pid, char state);
22: int FindPendingTask(Task& taskOut);
23: bool FindPendingMatch (int resources, unsigned int newPID, Task& taskOut);
24: unsigned int FindPID(const char* filename);
25: SDMComponent_ID GetPMAAddress(unsigned int pid);
26: void ClearList();
27: void PMFailure(const SDMComponent_ID& PMID);
28: void RemovePreferredPmId(const SDMTaskResources& TaskResources);
29: void PrintList();
30: Task& operator[](int index);
31: private:
32: Task tasks[NUMTASKS];
33: };
34:
35: #endif
```

File: sdm/common/task/SDMTaskResources.h

```
1: #ifndef __SDM_TASK_RESOURCES_H_
2: #define __SDM_TASK_RESOURCES_H_
3:
4: #include "taskdefs.h"
5: #include "../sdmLib.h"
6:
7: class SDMLIB_API SDMTaskResources
8: {
9: public:
10: SDMTaskResources();
11: SDMTaskResources(unsigned short usResources);
12: SDMTaskResources(const SDMTaskResources& other);
13:
14: void Set(unsigned short usResources);
15: void SetPreferredPmNodeId(unsigned short newPmId)
16: { m_usResources &= (ARCHMASK|MEMMASK|OSMASK); m_usResources |=
  PM_ID(newPmId); }
17: SDMTaskResources& operator=(const SDMTaskResources& other);
18:
19: bool IsPreferredPmIdSet() const { return ((m_usResources & PMIDMASK) != 0); }
20: unsigned short GetPreferredPmId() const { return (m_usResources & PMIDMASK); }
21: unsigned char GetPreferredPmIdNum() const { return ((m_usResources & PMIDMASK) >> 8); }
22: unsigned short GetArch() const { return (m_usResources & ARCHMASK); }
23: unsigned short GetMem() const { return (m_usResources & MEMMASK); }
24: unsigned short GetOs() const { return (m_usResources & OSMASK); }
25: unsigned short GetIgnorePmId() const { return (m_usResources &
  (ARCHMASK|MEMMASK|OSMASK)); }
26: unsigned short GetUShort() const { return m_usResources; }
27:
28: void SetArch(unsigned short usArch)
29: { m_usResources &= (PMIDMASK|MEMMASK|OSMASK); m_usResources|=
  (usArch&ARCHMASK); }
30: void SetMem(unsigned short usMem)
31: { m_usResources &= (PMIDMASK|ARCHMASK|OSMASK); m_usResources |=
  (usMem&MEMMASK); }
32: void SetOs(unsigned short usOs)
33: { m_usResources &= (PMIDMASK|ARCHMASK|MEMMASK); m_usResources |=
  (usOs&OSMASK); }
34:
35: static bool MatchResources (const SDMTaskResources& Pm, const SDMTaskResources& Task);
```

```
36: private:
37: unsigned short m_usResources;
38: };
39:
40: #endif
41:
```


File: sdm/common/task/tasklist.cpp

```
1: // TaskList implementation file
2:
3: #include "tasklist.h"
4: #include "../Exception/SDMBadIndexException.h"
5: #include "../message/SDMTaskError.h"
6: #include <string.h>
7: #include <stdio.h>
8:
9: TaskList::TaskList()
10: {   Init();
11: }
12:
13: void TaskList::Init()
14: {
15: for (int n=0; n<NUMTASKS; n++)
16: {   tasks[n].InitTask();
17: }
18: }
19:
20: bool TaskList::AnyInactive()
21: {
22: for (int n=0; n<NUMTASKS; n++)
23:     if (tasks[n].state == INACTIVE)
24:         return true;
25: return false;
26: }
27:
28: bool TaskList::AnyPending()
29: {
30: for (int n=0; n<NUMTASKS; n++)
31:     if (tasks[n].state == PENDING)
32:         return true;
33: return false;
34: }
35:
36: bool TaskList::AddTo(const Task& t)
37: {
38: int n;
39: if (!AnyInactive()) return false;
```

```

40: else
41: {
42:     for (n=0; n<NUMTASKS && tasks[n].state != INACTIVE; n++) ;
43:     if (n >= NUMTASKS) return false;
44:     else tasks[n] = t;
45: }
46: return true;
47: }
48:
49: bool TaskList::RemoveFrom(unsigned int p) // remove task by pid
50: {
51: int n;
52: for (n=0; n<NUMTASKS && tasks[n].pid != p; n++) ;
53: if (n >= NUMTASKS) return false;
54: else tasks[n].Delete();
55: return true;
56: }
57:
58: bool TaskList::RemoveFrom(char *filename)
59: {
60: int n;
61: if (filename == NULL)
62:     return false;
63: for (n=0; n<NUMTASKS; n++)
64: {
65:     if (tasks[n].filename != NULL)
66:         if (!strcmp(tasks[n].filename, filename))
67:             break;
68: }
69: if (n >= NUMTASKS)
70:     return false;
71: else
72:     tasks[n].Delete();
73: return true;
74: }
75:
76: /*
77: Perform whatever cleanup is needed when a PM reports that a task is finished.
78: Take the following actions depending on the task's mode:
79:  MODE_NORMAL -- The task is finished, remove it from the list.
80:  MODE_ALWAYS_RUNNING -- The task probably faulted, reschedule it.

```

```

81: */
82: bool TaskList::TaskFinished(unsigned int a_uiPid)
83: {
84:     int n = 0;
85:     bool bFound = false;
86:     bool bResult = false;
87:
88:     // Find the finished task
89:     for (n = 0; n < NUMTASKS; n++)
90:     {
91:         if (tasks[n].pid == a_uiPid)
92:         {
93:             bFound = true;
94:             break;
95:         }
96:     }
97:
98:     // If the task wasn't found, return error
99:     if (false == bFound)
100:     {
101:         return false;
102:     }
103:
104:     // Otherwise, n is the index of the task
105:     switch(tasks[n].taskMode)
106:     {
107:     case MODE_NORMAL:
108:         // Remove the task from the list
109:         bResult = RemoveFrom(a_uiPid);
110:         break;
111:     case MODE_ALWAYS_RUNNING:
112:         // Set the task to be rescheduled, remove its pid
113:         tasks[n].state = PENDING;
114:         tasks[n].pid = 0;
115:         bResult = true;
116:         break;
117:     default:
118:         printf("%s -- Invalid mode option. \n", __FUNCTION__);
119:         break;
120:     }
121:

```

```

122: return bResult;
123: }
124:
125: void TaskList::ClearList()
126: {
127:     for (int n = 0; n < NUMTASKS; n++)
128:     {
129:         tasks[n].Delete();
130:     }
131: }
132:
133: bool TaskList::IsFilePresent(char *f)
134: {
135:     int n;
136:     for (n=0; n<NUMTASKS; n++)
137:         if (tasks[n].filename != NULL)
138:             if (!strcmp(f, tasks[n].filename))
139:                 return true;
140:     return false;
141: }
142:
143: unsigned int TaskList::SetState (unsigned int p, char state)           // set state by pid
144: {
145:     int n;
146:     for (n=0; n<NUMTASKS; n++)
147:         if (tasks[n].pid == p)
148:         {   tasks[n].state = state;
149:             if (state == FINISHED)
150:                 tasks[n].pid = PID_INVALID;
151:             return tasks[n].pid;
152:         }
153:     return 0;
154: }
155:
156: /*bool MatchResources (int pmResources, int taskResources)           // verify   pmResources   <=
taskResources
157: {
158:     if ((taskResources & PMIDMASK) != 0)
159:         if ((pmResources & PMIDMASK) != (taskResources & PMIDMASK))
160:             return false;
161:

```

```

162:   if ((pmResources & ARCHMASK) == (taskResources & ARCHMASK))
163:       if ((pmResources & MEMMASK) >= (taskResources & MEMMASK))
164:           if ((pmResources & OSMASK) == (taskResources & OSMASK))
165:               return true;
166:   return false;
167: }*/
168:
169: // Find a pending task in the list
170: int TaskList::FindPendingTask(Task& taskOut)
171: {
172:   for (int n = 0; n < NUMTASKS; n++)
173:   {
174:       if (tasks[n].state == PENDING)
175:       {
176:           taskOut = tasks[n];
177:           return n;
178:       }
179:   }
180:   return -1;
181: }
182:
183: // Only match PENDING tasks (tasks that have not been scheduled)
184: bool TaskList::FindPendingMatch (int resources, unsigned int newPID, Task& taskOut)
185: {
186:   int n, lastpos;
187:   bool success = false;
188:   taskOut.InitTask();
189:   lastpos=0;
190:   for (n=0; n<NUMTASKS && !success; n++)
191:   {   if (tasks[n].state == PENDING)
192:       if (SDMTaskResources::MatchResources (resources, tasks[n].resources))
193:           if (tasks[n].priority >= taskOut.priority)
194:           {
195:               tasks[n].pid = newPID;
196:               taskOut = tasks[n];
197:               success = true;
198:               lastpos = n;
199:           }
200:   }
201:   return success;
202: }

```

```

203:
204: bool TaskList::SetAddress(unsigned int p, const SDMComponent_ID& PM)
205: {
206:     int n;
207:     bool found = false;
208:     for (n=0; n<NUMTASKS && !found; n++)
209:     {
210:         if (tasks[n].pid == p)
211:         {
212:             tasks[n].SetPM(PM);
213:             return true;
214:         }
215:     }
216:     return false;
217: }
218:
219: void TaskList::RemovePreferredPmId(const SDMTaskResources& TaskResources)
220: {
221:     for (int n = 0; n < NUMTASKS; n++)
222:     {
223:         if (tasks[n].state != INACTIVE && tasks[n].resources.GetPreferredPmId() ==
TaskResources.GetPreferredPmId())
224:             tasks[n].resources.SetPreferredPmNodeId(0);
225:     }
226: }
227:
228: unsigned int TaskList::FindPID(const char* filename)
229: {
230:     if (filename == NULL)
231:         return PID_INVALID;
232:     for (int n = 0; n < NUMTASKS; n++)
233:     {
234:         if (tasks[n].state == SCHEDULED && tasks[n].filename != NULL &&
strcmp(tasks[n].filename, filename)==0)
235:         {
236:             return tasks[n].pid;
237:         }
238:     }
239:     return PID_INVALID;
240: }
241:

```

```

242: // This PM has failed, set all tasks to PENDING and cancel their xTEDS
243: void TaskList::PMFailure(const SDMComponent_ID& PMID)
244: {
245:     SDMTaskError msgError;
246:     msgError.source = PMID;
247:     for (int n = 0; n < NUMTASKS; n++)
248:     {
249:         if (tasks[n].GetPM() == PMID)
250:         {
251:             msgError.source.setSensorID(tasks[n].pid);
252:             msgError.pid = tasks[n].pid;
253:             msgError.Send();
254:
255:             tasks[n].state = PENDING;
256:         }
257:     }
258: }
259:
260: void TaskList::PrintList()
261: {
262:     printf(" \n***Task List is: \n");
263:     for (int n = 0; n < NUMTASKS; n++)
264:     {
265:         if (tasks[n].state != INACTIVE)
266:         {
267:             char StateStr[16];
268:             switch(tasks[n].state)
269:             {
270:                 case INACTIVE:
271:                     strncpy(StateStr, "INACTIVE", sizeof(StateStr)); break;
272:                 case ACTIVE:
273:                     strncpy(StateStr, "ACTIVE", sizeof(StateStr)); break;
274:                 case ASSIGNED:
275:                     strncpy(StateStr, "ASSIGNED", sizeof(StateStr)); break;
276:                 case SCHEDULED:
277:                     strncpy(StateStr, "SCHEDULED", sizeof(StateStr)); break;
278:                 case PENDING:
279:                     strncpy(StateStr, "PENDING", sizeof(StateStr)); break;
280:                 case FINISHED:
281:                     strncpy(StateStr, "FINISHED", sizeof(StateStr)); break;
282:             }

```

```

283:         printf(" Index %d: name: %s pid: %d state: %s priority: %hd resources: %hd timeout:
%d mode: %d \n",n,tasks[n].filename, tasks[n].pid, StateStr, tasks[n].priority,
tasks[n].resources.GetUShort(), tasks[n].timeout, tasks[n].taskMode);
284:     }
285: }
286: printf ("*** \n \n");
287: }
288:
289: SDMComponent_ID TaskList::GetPMAddress(unsigned int pid)
290: {
291:     for (int n = 0; n < NUMTASKS; n++)
292:     {
293:         if (tasks[n].pid == pid)
294:         {
295:             return tasks[n].GetPM();
296:         }
297:     }
298:     return SDMComponent_ID();
299: }
300:
301: Task& TaskList::operator[] (int index)
302: {
303:     if (index < 0 || index > NUMTASKS)
304:         throw SDMBadIndexException(__FUNCTION__);
305:
306:     if (tasks[index].state == INACTIVE)
307:         throw SDMBadIndexException(__FUNCTION__);
308:
309:     return tasks[index];
310: }
311:

```


File: sdm/common/task/task.cpp

```
1: // Task classes implementation
2:
3: #include "task.h"
4: #include <stdlib.h>
5: #include <stdio.h>
6: #include <string.h>
7:
8: Task::Task():pmComponentID(),state('
                                \0'),priority('
                                \0'),pid(PID_INVALID),resources(0),filename(NULL),timeout(0),taskMode(MODE_NORMAL)
9: {
10: InitTask();
11: }
12:
13: Task::Task(const Task& a):pmComponentID(a.pmComponentID), state(a.state), priority(a.priority),
                             pid(a.pid), resources(a.resources),
                             filename(strdup(a.filename)),timeout(a.timeout),taskMode(a.taskMode)
14: {
15: }
16:
17: Task::~~Task()
18: {
19: if(filename != NULL)
20:     free(filename);
21: }
22:
23: void Task::InitTask()
24: {
25: resources = 0;
26: pid = PID_INVALID;
27: filename = NULL;
28: state = INACTIVE;
29: priority = 0;
30: pmComponentID.setAddress(0);
31: pmComponentID.setSensorID(0);
32: pmComponentID.setPort(0);
33: timeout = 0;
34: taskMode = MODE_NORMAL;
35: }
36:
37: void Task::Delete()
```

```

38: {
39: if (filename != NULL)
40:     free(filename);
41: InitTask();
42: }
43:
44: void Task::SetTask (char s, char pr, unsigned int p, const SDMTaskResources& r, char *f, int
atimeout, int aMode)
45: {
46: resources = r;
47: free (filename);
48: filename = (char *) malloc(strlen(f)+1);
49: strcpy (filename, f);
50: state = s;
51: pid = p;
52: priority = pr;
53: timeout = atimeout;
54: taskMode = aMode;
55: }
56:
57: Task& Task::operator= (const Task& t)
58: {
59: Delete();
60: SetTask (t.state, t.priority, t.pid, t.resources, t.filename, t.timeout, t.taskMode);
61: return *this;
62: }
63:
64: void Task::SetPM(const SDMComponent_ID& PM)
65: {
66: pmComponentID = PM;
67: }

```

File: sdm/common/task/task.h

```
1: // Task classes definitions      Cannon 1/05
2: #ifndef __TASK_H_
3: #define __TASK_H_
4:
5: #include <string.h>
6: #include "../sdmLib.h"
7: #include "../message/SDMComponent_ID.h"
8: #include "SDMTaskResources.h"
9: #include "taskdefs.h"
10:
11:
12: class SDMLIB_API Task
13: {
14: public:
15: Task();
16: Task(const Task&);
17: ~Task();
18: void InitTask ();
19: void Delete ();
20: void SetTask (char state, char priority, unsigned int pid, const SDMTaskResources& resources, char
    *filename, int timeout, int aMode);
21: Task& operator= (const Task& t);
22:
23: void SetPM(const SDMComponent_ID& PM);
24: SDMComponent_ID GetPM() const { return pmComponentID; }
25: public:
26: SDMComponent_ID pmComponentID;
27: char state;
28: char priority;
29: unsigned int pid;
30: SDMTaskResources resources;
31: char *filename;
32: int timeout;           // The timeout time needed to wait until the task should be scheduled
33: int taskMode;
34: };
35:
36: #endif
```

File: sdm/common/task/taskdefs.h

```
1: #ifndef _SDM_TASK_DEFS_H_
2: #define _SDM_TASK_DEFS_H_
3:
4: #define NUMTASKS50           // for pm
5: #define PID_INVALID  0
6:
7: // task status
8: #define INACTIVE      0
9: #define ACTIVE        1
10: #define ASSIGNED      2
11: #define SCHEDULED     3    // Assigned to a PM
12: #define PENDING       4    // Waiting to be scheduled
13: #define FINISHED      5    // Task state is finished running
14: #define RUNNING       6    // Task is running
15:
16: // task modes
17: #define MODE_NORMAL      0
18: #define MODE_ALWAYS_RUNNING  1
19:
20: // resource definitions
21: #define SDM_SPAU    0x0001    // first nibble: arch
22: #define SDM_INTEL   0x0002
23: #define SDM_PPC_7404 0x0004
24: #define SDM_PPC_755 0x0005
25: #define SDM_PPC_405 0x0006
26: #define SDM_MICROBLAZE 0x0007
27: #define SDM_SPARC  0x0008
28:
29: #define SDM_MEM32  0x0010    // second nibble: mem
30: #define SDM_MEM64  0x0020
31: #define SDM_MEM128 0x0030
32: #define SDM_MEM256 0x0040
33: #define SDM_MEM512 0x0050
34: #define SDM_MEM1024 0x0060
35:
36: #define SDM_LINUX24  0x0100    // third nibble: OS
37: #define SDM_LINUX26  0x0200
38: #define SDM_WIN32    0x0300
39: #define SDM_VXWORKS  0x0400
```

```
40:
41: #define ARCHMASK  0x000F      // architecture mask for resources
42: #define MEMMASK    0x00F0      // memory mask
43: #define OSMASK     0x0F00      // OS mask
44: #define PMIDMASK   0xF000      // PM node id mask
45:
46: // Used for specifying a pm node id in the resource requirements
47: // to be used in position a in 0xa000, must be between 1 and 255
48: #define PM_ID(value) (value << 12)
49:
50: // operating mode definitions
51: #define NORMAL_MODE  1
52: #define QUIET_MODE   2
53: #define SPECIAL_MODE 3
54: #define CRITICAL_MODE 4
55:
56: #endif
```

File: sdm/common/VarInfoParser/Variable.h

```
1: #ifndef __VARIABLE_H_
2: #define __VARIABLE_H_
3: #include "../xTEDS/xTEDSParser.h"
4: #include <stdlib.h>
5: /*
6:  *Utility functions for parsing and copying a variable definitions section of an xTEDS
7:  *The user should use the VarInfoParser object, not these functions.
8:  */
9:
10: variable* parseVarInfo(char *var_info);
11: variable* variableDeepCopy(variable* currVariable);
12: qualifier_type* qualifierDeepCopy(qualifier_type* currQualifier);
13: curve* curveDeepCopy(curve* currCurve);
14: coef* coefDeepCopy(coef* currCoef);
15: drange* drangeDeepCopy(drange* currDrange);
16: curveoption* optionDeepCopy(curveoption* currOption);
17:
18: #endif
```

File: sdm/common/VarInfoParser/Variable.c

```
1: #include <stdlib.h>
2: #include <string.h>
3: #include "Variable.h"
4:
5: /*
6:  * Invokes the variable parser and performs a deep copy of the variable object, freeing the one creating
  by the
7:  * parser and returning the pointer to the copied one.
8:  *   Params:
9:  *       var_info - The section of the xTEDS containing the variable definitions to parse.
10:  *   Returns:
11:  *       variable* - A pointer to the heap-allocated variable object, this must be deallocated by using
  delete_variable().
12:  */
13: variable* parseVarInfo(char *var_info)
14: {
15:     extern void* VarInfoParser_scan_string(const char *str);
16:     extern void VarInfoParser_delete_buffer(void*);
17:     extern int VarInfoParserparse();
18:     int parseResult = -1;
19:     /*Allocate a string buffer*/
20:     void *buffer = VarInfoParser_scan_string(var_info);
21:     /*Perform the parse*/
22:     parseResult = VarInfoParserparse();
23:     /*Free the string buffer*/
24:     VarInfoParser_delete_buffer(buffer);
25:     if (parseResult == 0)    /*If successful*/
26:     {
27:         variable *Copy = variableDeepCopy(getParsedVariable());
28:         delete_variable(getParsedVariable());
29:         return Copy;
30:     }
31:     return NULL;
32: }
33: /*
34:  * Performs a deep copy of a variable structure.
35:  *   Params:
36:  *       currVariable - The variable to copy.
37:  *   Returns:
```

```

38: *      variable* - The heap-allocated pointer of the variable object copied
39: */
40: variable* variableDeepCopy(variable* currVariable)
41: {
42: if (!currVariable)
43:     return NULL;
44: variable* Copy = (variable*) malloc (sizeof(variable));
45: memset(Copy, 0, sizeof(variable));
46: /*Copy (heap-allocated) all string fields to the new object*/
47: if (currVariable->length)    Copy->length = strdup(currVariable->length);
48: if (currVariable->kind) Copy->kind = strdup(currVariable->kind);
49: if (currVariable->name)    Copy->name = strdup(currVariable->name);
50: if (currVariable->qualifier) Copy->qualifier = strdup(currVariable->qualifier);
51: if (currVariable->id)      Copy->id = strdup(currVariable->id);
52: if (currVariable->range_min) Copy->range_min = strdup(currVariable->range_min);
53: if (currVariable->range_max) Copy->range_max = strdup(currVariable->range_max);
54: if (currVariable->default_value) Copy->default_value = strdup(currVariable->default_value);
55: if (currVariable->precision) Copy->precision = strdup(currVariable->precision);
56: if (currVariable->units) Copy->units = strdup(currVariable->units);
57: if (currVariable->accuracy) Copy->accuracy = strdup(currVariable->accuracy);
58: if (currVariable->scale_factor) Copy->scale_factor = strdup(currVariable->scale_factor);
59: if (currVariable->scale_units) Copy->scale_units = strdup(currVariable->scale_units);
60: if (currVariable->format) Copy->format = strdup(currVariable->format);
61: if (currVariable->description) Copy->description = strdup(currVariable->description);
62: if (currVariable->interface_name) Copy->interface_name = strdup(currVariable->interface_name);
63: if (currVariable->interface_id) Copy->interface_id = strdup(currVariable->interface_id);
64: /*Copy the qualifiers*/
65: Copy->qualifiers = qualifierDeepCopy(currVariable->qualifiers);
66: /*Copy the curves*/
67: Copy->curves = curveDeepCopy(currVariable->curves);
68: /*Copy the Dranges*/
69: Copy->dranges = drangeDeepCopy(currVariable->dranges);
70: /*Copy the next variable*/
71: Copy->next = variableDeepCopy(currVariable->next);
72: return Copy;
73: }
74: /*
75: * Performs a deep copy of a qualifier_type structure.
76: * Params:
77: *      currQualifier - The qualifier_type to copy.
78: * Returns:

```



```

79: *      qualifier_type* - The heap-allocated pointer of the qualifier_type object copied
80: */
81: qualifier_type* qualifierDeepCopy(qualifier_type* currQualifier)
82: {
83: if (!currQualifier)
84:     return NULL;
85: qualifier_type* Copy = (qualifier_type*) malloc (sizeof(qualifier_type));
86: memset(Copy, 0, sizeof(qualifier_type));
87: /*Copy (heap-allocated) all fields to the new object*/
88: if (currQualifier->name) Copy->name = strdup(currQualifier->name);
89: if (currQualifier->value) Copy->value = strdup(currQualifier->value);
90: if (currQualifier->units) Copy->units = strdup(currQualifier->units);
91: /*Copy the next qualifier*/
92: Copy->next = qualifierDeepCopy(currQualifier->next);
93: return Copy;
94: }
95: /*
96: * Performs a deep copy of a curve structure.
97: * Params:
98: *      currCurve - The curve to copy.
99: * Returns:
100: *      curve* - The heap-allocated pointer of the curve object copied
101: */
102: curve* curveDeepCopy(curve* currCurve)
103: {
104: if (!currCurve)
105:     return NULL;
106: curve* Copy = (curve*) malloc(sizeof(curve));
107: memset(Copy, 0, sizeof(curve));
108: /*Copy (heap-allocated) all fields to the new object*/
109: if (currCurve->name) Copy->name = strdup(currCurve->name);
110: if (currCurve->description) Copy->description = strdup(currCurve->description);
111: /*Copy the coefficient data*/
112: Copy->coefs = coefDeepCopy(currCurve->coefs);
113: return Copy;
114: }
115: /*
116: * Performs a deep copy of a coef structure.
117: * Params:
118: *      currCoef - The coef to copy.
119: * Returns:

```

```

120: *      variable* - The heap-allocated pointer of the coef object copied
121: */
122: coef* coefDeepCopy(coef* currCoef)
123: {
124:     if (!currCoef)
125:         return NULL;
126:     coef* Copy = (coef*) malloc(sizeof(coef));
127:     memset(Copy, 0, sizeof(coef));
128:     /*Copy (heap-allocated) all fields to the new object*/
129:     if (currCoef->exponent) Copy->exponent = strdup(currCoef->exponent);
130:     if (currCoef->value) Copy->value = strdup(currCoef->value);
131:     if (currCoef->description) Copy->description = strdup(currCoef->description);
132:     /*Copy the next coef*/
133:     Copy->next = coefDeepCopy(currCoef->next);
134:     return Copy;
135: }
136: /*
137: * Performs a deep copy of a drange structure.
138: * Params:
139: *      currDrange - The drange to copy.
140: * Returns:
141: *      variable* - The heap-allocated pointer of the drange object copied
142: */
143: drange* drangeDeepCopy(drange* currDrange)
144: {
145:     if (!currDrange)
146:         return NULL;
147:     drange* Copy = (drange*) malloc(sizeof(drange));
148:     memset(Copy, 0, sizeof(drange));
149:     /*Copy (heap-allocated) all fields to the new object*/
150:     if (currDrange->name) Copy->name = strdup(currDrange->name);
151:     if (currDrange->description) Copy->description = strdup(currDrange->description);
152:     /*Copy the options*/
153:     Copy->options = optionDeepCopy(currDrange->options);
154:     return Copy;
155: }
156: /*
157: * Performs a deep copy of an option structure.
158: * Params:
159: *      currOption - The option to copy.
160: * Returns:

```

```

161: *      option* - The heap-allocated pointer of the option object copied
162: */
163: curveoption* optionDeepCopy(curveoption* currOption)
164: {
165:     if (!currOption)
166:         return NULL;
167:     curveoption* Copy = (curveoption*) malloc(sizeof(curveoption));
168:     memset(Copy, 0, sizeof(curveoption));
169:     /*Copy (heap-allocated) all fields to the new object*/
170:     if (currOption->name) Copy->name = strdup(currOption->name);
171:     if (currOption->value) Copy->value = strdup(currOption->value);
172:     if (currOption->description) Copy->description = strdup(currOption->description);
173:     if (currOption->alarm) Copy->alarm = strdup(currOption->alarm);
174:     /*Copy the next option*/
175:     Copy->next = optionDeepCopy(currOption->next);
176:     return Copy;
177: }

```

File: sdm/common/VarInfoParser/lex.VarInfoParser.c

```
1: #define yy_create_buffer VarInfoParser_create_buffer
2: #define yy_delete_buffer VarInfoParser_delete_buffer
3: #define yy_scan_buffer VarInfoParser_scan_buffer
4: #define yy_scan_string VarInfoParser_scan_string
5: #define yy_scan_bytes VarInfoParser_scan_bytes
6: #define yy_flex_debug VarInfoParser_flex_debug
7: #define yy_init_buffer VarInfoParser_init_buffer
8: #define yy_flush_buffer VarInfoParser_flush_buffer
9: #define yy_load_buffer_state VarInfoParser_load_buffer_state
10: #define yy_switch_to_buffer VarInfoParser_switch_to_buffer
11: #define yyin VarInfoParserin
12: #define yyleng VarInfoParserleng
13: #define yylex VarInfoParserlex
14: #define yyout VarInfoParserout
15: #define yyrestart VarInfoParserrestart
16: #define yytext VarInfoParsertext
17: #define yywrap VarInfoParserwrap
18:
19: /* A lexical scanner generated by flex*/
20:
21: /* Scanner skeleton version:
22:  * $Header: /home/daffy/u0/vern/flex/RCS/flex.skl,v 2.91 96/09/10 16:58:48 vern Exp $
23:  */
24:
25: #define FLEX_SCANNER
26: #define YY_FLEX_MAJOR_VERSION 2
27: #define YY_FLEX_MINOR_VERSION 5
28:
29: #include <stdio.h>
30: #include <unistd.h>
31:
32:
33: /* cfront 1.2 defines "c_plusplus" instead of "__cplusplus" */
34: #ifdef c_plusplus
35: #ifndef __cplusplus
36: #define __cplusplus
37: #endif
38: #endif
39:
```

```

40:
41: #ifdef __cplusplus
42:
43: #include <stdlib.h>
44:
45: /* Use prototypes in function declarations. */
46: #define YY_USE_PROTOS
47:
48: /* The "const" storage-class-modifier is valid. */
49: #define YY_USE_CONST
50:
51: #else    /* ! __cplusplus */
52:
53: #if __STDC__
54:
55: #define YY_USE_PROTOS
56: #define YY_USE_CONST
57:
58: #endif  /* __STDC__ */
59: #endif /* ! __cplusplus */
60:
61: #ifdef __TURBOC__
62: #pragma warn -rch
63: #pragma warn -use
64: #include <io.h>
65: #include <stdlib.h>
66: #define YY_USE_CONST
67: #define YY_USE_PROTOS
68: #endif
69:
70: #ifdef YY_USE_CONST
71: #define yyconst const
72: #else
73: #define yyconst
74: #endif
75:
76:
77: #ifdef YY_USE_PROTOS
78: #define YY_PROTO(proto) proto
79: #else
80: #define YY_PROTO(proto) ()

```

```

81: #endif
82:
83: /* Returned upon end-of-file. */
84: #define YY_NULL 0
85:
86: /* Promotes a possibly negative, possibly signed char to an unsigned
87:  * integer for use as an array index.  If the signed char is negative,
88:  * we want to instead treat it as an 8-bit unsigned char, hence the
89:  * double cast.
90:  */
91: #define YY_SC_TO_UI(c) ((unsigned int) (unsigned char) c)
92:
93: /* Enter a start condition.  This macro really ought to take a parameter,
94:  * but we do it the disgusting crufty way forced on us by the ()-less
95:  * definition of BEGIN.
96:  */
97: #define BEGIN yy_start = 1 + 2 *
98:
99: /* Translate the current start state into a value that can be later handed
100:  * to BEGIN to return to the state.  The YYSTATE alias is for lex
101:  * compatibility.
102:  */
103: #define YY_START ((yy_start - 1) / 2)
104: #define YYSTATE YY_START
105:
106: /* Action number for EOF rule of a given start state. */
107: #define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
108:
109: /* Special action meaning "start processing a new file". */
110: #define YY_NEW_FILE yyrestart( yyin )
111:
112: #define YY_END_OF_BUFFER_CHAR 0
113:
114: /* Size of default input buffer. */
115: #define YY_BUF_SIZE 16384
116:
117: typedef struct yy_buffer_state *YY_BUFFER_STATE;
118:
119: extern int yyleng;
120: extern FILE *yyin, *yyout;
121:

```

```

122: #define EOB_ACT_CONTINUE_SCAN 0
123: #define EOB_ACT_END_OF_FILE 1
124: #define EOB_ACT_LAST_MATCH 2
125:
126: /* The funky do-while in the following #define is used to turn the definition
127:  * int a single C statement (which needs a semi-colon terminator). This
128:  * avoids problems with code like:
129:  *
130:  * if ( condition_holds )
131:  *     yyless( 5 );
132:  * else
133:  *     do_something_else();
134:  *
135:  * Prior to using the do-while the compiler would get upset at the
136:  * "else" because it interpreted the "if" statement as being all
137:  * done when it reached the ';' after the yyless() call.
138:  */
139:
140: /* Return all but the first 'n' matched characters back to the input stream. */
141:
142: #define yyless(n) \
143:     do \
144:         { \
145:             /* Undo effects of setting up yytext. */ \
146:             *yy_cp = yy_hold_char; \
147:             YY_RESTORE_YY_MORE_OFFSET \
148:             yy_c_buf_p = yy_cp = yy_bp + n - YY_MORE_ADJ; \
149:             YY_DO_BEFORE_ACTION; /* set up yytext again */ \
150:         } \
151:     while ( 0 )
152:
153: #define unput(c) yyunput( c, yytext_ptr )
154:
155: /* Some routines like yy_flex_realloc() are emitted as static but are
156:  not called by all lexers. This generates warnings in some compilers,
157:  notably GCC. Arrange to suppress these. */
158: #ifdef __GNUC__
159: #define YY_MAY_BE_UNUSED __attribute__((unused))
160: #else
161: #define YY_MAY_BE_UNUSED
162: #endif

```

```

163:
164: /* The following is because we cannot portably get our hands on size_t
165:  * (without autoconf's help, which isn't available because we want
166:  * flex-generated scanners to compile on their own).
167:  */
168: typedef unsigned int yy_size_t;
169:
170:
171: struct yy_buffer_state
172: {
173:     FILE *yy_input_file;
174:
175:     char *yy_ch_buf;      /* input buffer */
176:     char *yy_buf_pos;     /* current position in input buffer */
177:
178:     /* Size of input buffer in bytes, not including room for EOB
179:      * characters.
180:      */
181:     yy_size_t yy_buf_size;
182:
183:     /* Number of characters read into yy_ch_buf, not including EOB
184:      * characters.
185:      */
186:     int yy_n_chars;
187:
188:     /* Whether we "own" the buffer - i.e., we know we created it,
189:      * and can realloc() it to grow it, and should free() it to
190:      * delete it.
191:      */
192:     int yy_is_our_buffer;
193:
194:     /* Whether this is an "interactive" input source; if so, and
195:      * if we're using stdio for input, then we want to use getc()
196:      * instead of fread(), to make sure we stop fetching input after
197:      * each newline.
198:      */
199:     int yy_is_interactive;
200:
201:     /* Whether we're considered to be at the beginning of a line.
202:      * If so, '^' rules will be active on the next match, otherwise
203:      * not.

```



```

204:  */
205:  int yy_at_bol;
206:
207:  /* Whether to try to fill the input buffer when we reach the
208:   * end of it.
209:   */
210:  int yy_fill_buffer;
211:
212:  int yy_buffer_status;
213: #define YY_BUFFER_NEW 0
214: #define YY_BUFFER_NORMAL 1
215:  /* When an EOF's been seen but there's still some text to process
216:   * then we mark the buffer as YY_EOF_PENDING, to indicate that we
217:   * shouldn't try reading from the input source any more.  We might
218:   * still have a bunch of tokens to match, though, because of
219:   * possible backing-up.
220:   *
221:   * When we actually see the EOF, we change the status to "new"
222:   * (via yyrestart()), so that the user can continue scanning by
223:   * just pointing yyin at a new input file.
224:   */
225: #define YY_BUFFER_EOF_PENDING 2
226: };
227:
228: static YY_BUFFER_STATE yy_current_buffer = 0;
229:
230: /* We provide macros for accessing buffer states in case in the
231:  * future we want to put the buffer states in a more general
232:  * "scanner state".
233:  */
234: #define YY_CURRENT_BUFFER yy_current_buffer
235:
236:
237: /* yy_hold_char holds the character lost when yytext is formed. */
238: static char yy_hold_char;
239:
240: static int yy_n_chars;      /* number of characters read into yy_ch_buf */
241:
242:
243: int yyleng;
244:

```

```

245: /* Points to current character in buffer. */
246: static char *yy_c_buf_p = (char *) 0;
247: static int yy_init = 1;      /* whether we need to initialize */
248: static int yy_start = 0; /* start state number */
249:
250: /* Flag which is used to allow yywrap()'s to do buffer switches
251:  * instead of setting up a fresh yyin.  A bit of a hack ...
252:  */
253: static int yy_did_buffer_switch_on_eof;
254:
255: void yyrestart YY_PROTO(( FILE *input_file ));
256:
257: void yy_switch_to_buffer YY_PROTO(( YY_BUFFER_STATE new_buffer ));
258: void yy_load_buffer_state YY_PROTO(( void ));
259: YY_BUFFER_STATE yy_create_buffer YY_PROTO(( FILE *file, int size ));
260: void yy_delete_buffer YY_PROTO(( YY_BUFFER_STATE b ));
261: void yy_init_buffer YY_PROTO(( YY_BUFFER_STATE b, FILE *file ));
262: void yy_flush_buffer YY_PROTO(( YY_BUFFER_STATE b ));
263: #define YY_FLUSH_BUFFER yy_flush_buffer( yy_current_buffer )
264:
265: YY_BUFFER_STATE yy_scan_buffer YY_PROTO(( char *base, yy_size_t size ));
266: YY_BUFFER_STATE yy_scan_string YY_PROTO(( yyconst char *yy_str ));
267: YY_BUFFER_STATE yy_scan_bytes YY_PROTO(( yyconst char *bytes, int len ));
268:
269: static void *yy_flex_alloc YY_PROTO(( yy_size_t ));
270: static void *yy_flex_realloc YY_PROTO(( void *, yy_size_t )) YY_MAY_BE_UNUSED;
271: static void yy_flex_free YY_PROTO(( void * ));
272:
273: #define yy_new_buffer yy_create_buffer
274:
275: #define yy_set_interactive(is_interactive) \
276:   { \
277:     if ( ! yy_current_buffer ) \
278:       yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE ); \
279:     yy_current_buffer->yy_is_interactive = is_interactive; \
280:   }
281:
282: #define yy_set_bol(at_bol) \
283:   { \
284:     if ( ! yy_current_buffer ) \
285:       yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE ); \

```

```

286: yy_current_buffer->yy_at_bol = at_bol; \
287: }
288:
289: #define YY_AT_BOL() (yy_current_buffer->yy_at_bol)
290:
291: typedef unsigned char YY_CHAR;
292: FILE *yyin = (FILE *) 0, *yyout = (FILE *) 0;
293: typedef int yy_state_type;
294: extern char *yytext;
295: #define yytext_ptr yytext
296:
297: static yy_state_type yy_get_previous_state YY_PROTO(( void ));
298: static yy_state_type yy_try_NUL_trans YY_PROTO(( yy_state_type current_state ));
299: static int yy_get_next_buffer YY_PROTO(( void ));
300: static void yy_fatal_error YY_PROTO(( yyconst char msg[] ));
301:
302: /* Done after the current pattern has been matched and before the
303:  * corresponding action - sets up yytext.
304:  */
305: #define YY_DO_BEFORE_ACTION \
306:   yytext_ptr = yy_bp; \
307:   yyleng = (int) (yy_cp - yy_bp); \
308:   yy_hold_char = *yy_cp; \
309:   *yy_cp = '\0'; \
310:   yy_c_buf_p = yy_cp;
311:
312: #define YY_NUM_RULES 34
313: #define YY_END_OF_BUFFER 35
314: static yyconst short int yy_accept[201] =
315: { 0,
316:   32, 32, 35, 33, 32, 32, 33, 33, 33, 30,
317:   33, 1, 2, 33, 33, 33, 33, 33, 33, 33,
318:   33, 33, 33, 33, 33, 33, 32, 0, 28, 29,
319:   3, 0, 30, 0, 0, 0, 0, 0, 0, 0,
320:   0, 0, 14, 0, 0, 0, 0, 0, 0, 0,
321:   0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
322:   0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
323:   0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
324:   0, 0, 0, 0, 0, 0, 0, 13, 0, 12,
325:   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
326:

```

```

327:    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
328:    0, 0, 0, 0, 0, 0, 0, 23, 27, 0,
329:    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
330:    0, 0, 0, 0, 0, 0, 19, 26, 0, 0,
331:    0, 0, 0, 0, 31, 0, 0, 0, 0, 6,
332:    8, 0, 0, 0, 0, 0, 0, 0, 0, 0,
333:    0, 0, 7, 9, 0, 0, 0, 0, 20, 0,
334:    0, 0, 0, 18, 21, 0, 0, 0, 0, 0,
335:    4, 0, 0, 17, 15, 0, 0, 0, 5, 10,
336:    0, 0, 0, 25, 11, 0, 16, 22, 24, 0
337:
338:    };
339:
340: static yyconst int yy_ec[256] =
341:    { 0,
342:      1, 1, 1, 1, 1, 1, 1, 1, 2, 3,
343:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
344:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
345:      1, 4, 5, 6, 1, 1, 1, 1, 1, 1,
346:      1, 1, 1, 1, 7, 8, 9, 10, 10, 10,
347:      10, 10, 10, 10, 10, 10, 10, 1, 1, 11,
348:      12, 13, 1, 1, 1, 1, 1, 14, 1, 15,
349:      1, 1, 1, 1, 1, 1, 16, 1, 17, 1,
350:      18, 1, 1, 1, 19, 20, 1, 1, 1, 1,
351:      1, 1, 1, 1, 1, 1, 21, 22, 23, 24,
352:
353:      25, 26, 27, 28, 29, 1, 30, 31, 32, 33,
354:      34, 35, 36, 37, 38, 39, 40, 41, 1, 42,
355:      43, 1, 1, 1, 1, 1, 1, 1, 1, 1,
356:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
357:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
358:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
359:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
360:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
361:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
362:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
363:
364:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
365:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
366:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
367:      1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

```

```

368:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
369:    1, 1, 1, 1, 1
370: };
371:
372: static yyconst int yy_meta[44] =
373: { 0,
374:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
375:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
376:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
377:    1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
378:    1, 1, 1
379: };
380:
381: static yyconst short int yy_base[205] =
382: { 0,
383:    0, 0, 241, 242, 42, 45, 234, 229, 225, 42,
384:    46, 242, 242, 214, 211, 201, 210, 204, 207, 210,
385:    193, 189, 207, 204, 193, 204, 54, 218, 242, 213,
386:    242, 212, 51, 214, 51, 183, 184, 178, 196, 193,
387:    36, 178, 242, 181, 180, 180, 186, 189, 176, 187,
388:    178, 175, 198, 167, 168, 162, 180, 179, 160, 177,
389:    160, 156, 174, 171, 161, 168, 164, 165, 166, 157,
390:    160, 155, 146, 144, 69, 162, 143, 160, 143, 146,
391:    149, 146, 147, 138, 134, 136, 151, 242, 132, 242,
392:    141, 140, 143, 142, 128, 140, 72, 74, 75, 157,
393:
394:    130, 133, 130, 131, 132, 124, 128, 135, 134, 123,
395:    124, 113, 123, 112, 123, 132, 65, 242, 242, 80,
396:    82, 140, 139, 83, 132, 117, 109, 113, 120, 115,
397:    106, 112, 115, 113, 96, 99, 242, 242, 104, 103,
398:    67, 110, 97, 116, 242, 103, 94, 100, 103, 242,
399:    242, 95, 92, 79, 101, 81, 85, 93, 75, 83,
400:    92, 85, 242, 242, 84, 81, 86, 85, 242, 88,
401:    79, 74, 69, 242, 242, 66, 65, 78, 77, 64,
402:    242, 69, 65, 242, 242, 64, 59, 58, 242, 242,
403:    54, 60, 55, 242, 242, 66, 242, 242, 242, 242,
404:
405:    69, 66, 53, 52
406: };
407:
408: static yyconst short int yy_def[205] =

```

```

409: { 0,
410:   200, 1, 200, 200, 200, 200, 201, 200, 200, 200,
411:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
412:   200, 200, 200, 200, 200, 200, 200, 201, 200, 200,
413:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
414:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
415:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
416:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
417:   200, 200, 200, 200, 202, 200, 200, 200, 200, 200,
418:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
419:   200, 200, 200, 200, 200, 200, 203, 202, 203, 204,
420:
421:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
422:   200, 200, 200, 200, 200, 200, 200, 200, 200, 203,
423:   202, 204, 204, 202, 200, 200, 200, 200, 200, 200,
424:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
425:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
426:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
427:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
428:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
429:   200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
430:   200, 200, 200, 200, 200, 200, 200, 200, 200, 0,
431:
432:   200, 200, 200, 200
433: };
434:
435: static yyconst short int yy_nxt[286] =
436: { 0,
437:   4, 5, 6, 5, 4, 7, 4, 8, 9, 10,
438:   11, 12, 13, 4, 4, 4, 4, 4, 4, 4,
439:   14, 4, 4, 15, 4, 16, 4, 4, 17, 18,
440:   19, 4, 20, 4, 21, 22, 23, 24, 4, 25,
441:   26, 4, 4, 27, 27, 27, 27, 27, 27, 32,
442:   34, 33, 124, 120, 35, 27, 27, 27, 32, 36,
443:   33, 63, 37, 38, 54, 39, 97, 55, 56, 28,
444:   57, 98, 99, 64, 121, 100, 98, 121, 122, 142,
445:   122, 123, 121, 143, 98, 98, 100, 159, 100, 100,
446:   199, 198, 197, 196, 195, 160, 194, 193, 192, 191,
447:
448:   190, 189, 188, 187, 186, 185, 184, 183, 182, 181,
449:   180, 179, 178, 177, 176, 175, 174, 173, 172, 171,

```

```

450: 170, 169, 168, 167, 166, 165, 164, 163, 145, 162,
451: 161, 158, 157, 156, 155, 154, 153, 152, 151, 150,
452: 149, 148, 147, 146, 145, 144, 200, 141, 140, 139,
453: 138, 137, 136, 135, 134, 133, 132, 131, 130, 129,
454: 128, 127, 126, 125, 119, 118, 117, 116, 115, 114,
455: 113, 112, 111, 110, 109, 108, 107, 106, 105, 104,
456: 103, 102, 101, 96, 95, 94, 93, 92, 91, 90,
457: 89, 88, 87, 86, 85, 84, 83, 82, 81, 80,
458:
459: 79, 78, 77, 76, 75, 74, 73, 72, 71, 70,
460: 69, 68, 67, 66, 65, 62, 61, 60, 59, 58,
461: 53, 30, 30, 29, 52, 51, 50, 49, 48, 47,
462: 46, 45, 44, 43, 42, 41, 40, 31, 30, 29,
463: 200, 3, 200, 200, 200, 200, 200, 200, 200, 200,
464: 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
465: 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
466: 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
467: 200, 200, 200, 200, 200
468: };
469:
470: static yyconst short int yy_chk[286] =
471: { 0,
472: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
473: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
474: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
475: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
476: 1, 1, 1, 5, 5, 5, 6, 6, 6, 10,
477: 11, 10, 204, 203, 11, 27, 27, 27, 33, 11,
478: 33, 41, 11, 11, 35, 11, 202, 35, 35, 201,
479: 35, 75, 75, 41, 97, 75, 98, 99, 97, 117,
480: 98, 99, 120, 117, 121, 124, 120, 141, 121, 124,
481: 196, 193, 192, 191, 188, 141, 187, 186, 183, 182,
482:
483: 180, 179, 178, 177, 176, 173, 172, 171, 170, 168,
484: 167, 166, 165, 162, 161, 160, 159, 158, 157, 156,
485: 155, 154, 153, 152, 149, 148, 147, 146, 144, 143,
486: 142, 140, 139, 136, 135, 134, 133, 132, 131, 130,
487: 129, 128, 127, 126, 125, 123, 122, 116, 115, 114,
488: 113, 112, 111, 110, 109, 108, 107, 106, 105, 104,
489: 103, 102, 101, 100, 96, 95, 94, 93, 92, 91,
490: 89, 87, 86, 85, 84, 83, 82, 81, 80, 79,

```

```

491:    78, 77, 76, 74, 73, 72, 71, 70, 69, 68,
492:    67, 66, 65, 64, 63, 62, 61, 60, 59, 58,
493:
494:    57, 56, 55, 54, 53, 52, 51, 50, 49, 48,
495:    47, 46, 45, 44, 42, 40, 39, 38, 37, 36,
496:    34, 32, 30, 28, 26, 25, 24, 23, 22, 21,
497:    20, 19, 18, 17, 16, 15, 14, 9, 8, 7,
498:    3, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
499:    200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
500:    200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
501:    200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
502:    200, 200, 200, 200, 200
503:    };
504:
505: static yy_state_type yy_last_accepting_state;
506: static char *yy_last_accepting_cpos;
507:
508: /* The intent behind this definition is that it'll catch
509:  * any uses of REJECT which flex missed.
510:  */
511: #define REJECT reject_used_but_not_detected
512: #define yymore() yymore_used_but_not_detected
513: #define YY_MORE_ADJ 0
514: #define YY_RESTORE_YY_MORE_OFFSET
515: char *yytext;
516: #line 1 "VarInfoParser.l"
517: #define INITIAL 0
518: #line 2 "VarInfoParser.l"
519: #define _GNU_SOURCE
520: #include <stdio.h>
521: #include <stdlib.h>
522: #include <string.h>
523:
524: #ifndef WIN32
525: #ifndef __VXWORKS__
526:     #ifndef strndup
527:         #define strndup(s,n) SDM_strndup(s,n)
528:     #endif
529: #endif
530: #else
531: #include "unistd.h"

```



```

532: #endif
533:
534: #include "VarInfoParser.tab.h"
535: #line 534 "lex.VarInfoParser.c"
536:
537: /* Macros after this point can all be overridden by user definitions in
538:  * section 1.
539:  */
540:
541: #ifndef YY_SKIP_YYWRAP
542: #ifdef __cplusplus
543: extern "C" int yywrap YY_PROTO(( void ));
544: #else
545: extern int yywrap YY_PROTO(( void ));
546: #endif
547: #endif
548:
549: #ifndef YY_NO_UNPUT
550: static void yyunput YY_PROTO(( int c, char *buf_ptr ));
551: #endif
552:
553: #ifndef yytext_ptr
554: static void yy_flex_strncpy YY_PROTO(( char *, yyconst char *, int ));
555: #endif
556:
557: #ifdef YY_NEED_STRLEN
558: static int yy_flex_strlen YY_PROTO(( yyconst char * ));
559: #endif
560:
561: #ifndef YY_NO_INPUT
562: #ifdef __cplusplus
563: static int yyinput YY_PROTO(( void ));
564: #else
565: static int input YY_PROTO(( void ));
566: #endif
567: #endif
568:
569: #if YY_STACK_USED
570: static int yy_start_stack_ptr = 0;
571: static int yy_start_stack_depth = 0;
572: static int *yy_start_stack = 0;

```

```

573: #ifndef YY_NO_PUSH_STATE
574: static void yy_push_state YY_PROTO(( int new_state ));
575: #endif
576: #ifndef YY_NO_POP_STATE
577: static void yy_pop_state YY_PROTO(( void ));
578: #endif
579: #ifndef YY_NO_TOP_STATE
580: static int yy_top_state YY_PROTO(( void ));
581: #endif
582:
583: #else
584: #define YY_NO_PUSH_STATE 1
585: #define YY_NO_POP_STATE 1
586: #define YY_NO_TOP_STATE 1
587: #endif
588:
589: #ifdef YY_MALLOC_DECL
590: YY_MALLOC_DECL
591: #else
592: #if __STDC__
593: #ifndef __cplusplus
594: #include <stdlib.h>
595: #endif
596: #else
597: /* Just try to get by without declaring the routines.  This will fail
598:  * miserably on non-ANSI systems for which sizeof(size_t) != sizeof(int)
599:  * or sizeof(void*) != sizeof(int).
600:  */
601: #endif
602: #endif
603:
604: /* Amount of stuff to slurp up with each read. */
605: #ifndef YY_READ_BUF_SIZE
606: #define YY_READ_BUF_SIZE 8192
607: #endif
608:
609: /* Copy whatever the last rule matched to the standard output. */
610:
611: #ifndef ECHO
612: /* This used to be an fputs(), but since the string might contain NUL's,
613:  * we now use fwrite().

```

```

614: */
615: #define ECHO (void) fwrite( yytext, yyleng, 1, yyout )
616: #endif
617:
618: /* Gets input and stuffs it into "buf".  number of characters read, or YY_NULL,
619:  * is returned in "result".
620: */
621: #ifndef YY_INPUT
622: #define YY_INPUT(buf,result,max_size) \
623:   if ( yy_current_buffer->yy_is_interactive ) \
624:       { \
625:         int c = '*'; n; \
626:         for ( n = 0; n < max_size && \
627:             (c = getc( yyin )) != EOF && c != '\n'; ++n ) \
628:             buf[n] = (char) c; \
629:         if ( c == '\n' ) \
630:             buf[n++] = (char) c; \
631:         if ( c == EOF && ferror( yyin ) ) \
632:             YY_FATAL_ERROR( "input in flex scanner failed" ); \
633:         result = n; \
634:       } \
635:   else if ( ((result = fread( buf, 1, max_size, yyin )) == 0) \
636:       && ferror( yyin ) ) \
637:       YY_FATAL_ERROR( "input in flex scanner failed" );
638: #endif
639:
640: /* No semi-colon after return; correct usage is to write "yyterminate();" -
641:  * we don't want an extra ';' after the "return" because that will cause
642:  * some compilers to complain about unreachable statements.
643: */
644: #ifndef yyterminate
645: #define yyterminate() return YY_NULL
646: #endif
647:
648: /* Number of entries by which start-condition stack grows. */
649: #ifndef YY_START_STACK_INCR
650: #define YY_START_STACK_INCR 25
651: #endif
652:
653: /* Report a fatal error. */
654: #ifndef YY_FATAL_ERROR

```

```

655: #define YY_FATAL_ERROR(msg) yy_fatal_error( msg )
656: #endif
657:
658: /* Default declaration of generated scanner - a define so the user can
659:  * easily add parameters.
660:  */
661: #ifndef YY_DECL
662: #define YY_DECL int yylex YY_PROTO(( void ))
663: #endif
664:
665: /* Code executed at the beginning of each rule, after yytext and yyleng
666:  * have been set up.
667:  */
668: #ifndef YY_USER_ACTION
669: #define YY_USER_ACTION
670: #endif
671:
672: /* Code executed at the end of each rule. */
673: #ifndef YY_BREAK
674: #define YY_BREAK break;
675: #endif
676:
677: #define YY_RULE_SETUP \
678:     YY_USER_ACTION
679:
680: YY_DECL
681: {
682:     register yy_state_type yy_current_state;
683:     register char *yy_cp = NULL, *yy_bp = NULL;
684:     register int yy_act;
685:
686: #line 18 "VarInfoParser.l"
687:
688:
689: #line 688 "lex.VarInfoParser.c"
690:
691:     if ( yy_init )
692:     {
693:         yy_init = 0;
694:
695: #ifdef YY_USER_INIT

```

```

696:     YY_USER_INIT;
697: #endif
698:
699:     if ( ! yy_start )
700:         yy_start = 1;    /* first start state */
701:
702:     if ( ! yyin )
703:         yyin = stdin;
704:
705:     if ( ! yyout )
706:         yyout = stdout;
707:
708:     if ( ! yy_current_buffer )
709:         yy_current_buffer =
710:             yy_create_buffer( yyin, YY_BUF_SIZE );
711:
712:     yy_load_buffer_state();
713: }
714:
715: while ( 1 )    /* loops until end-of-file is reached */
716: {
717:     yy_cp = yy_c_buf_p;
718:
719:     /* Support of yytext. */
720:     *yy_cp = yy_hold_char;
721:
722:     /* yy_bp points to the position in yy_ch_buf of the start of
723:      * the current run.
724:      */
725:     yy_bp = yy_cp;
726:
727:     yy_current_state = yy_start;
728: yy_match:
729:     do
730:     {
731:         register YY_CHAR yy_c = yy_ec[YY_SC_TO_UI(*yy_cp)];
732:         if ( yy_accept[yy_current_state] )
733:         {
734:             yy_last_accepting_state = yy_current_state;
735:             yy_last_accepting_cpos = yy_cp;
736:         }

```

```

737:         while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
738:             {
739:                 yy_current_state = (int) yy_def[yy_current_state];
740:                 if ( yy_current_state >= 201 )
741:                     yy_c = yy_meta[(unsigned int) yy_c];
742:             }
743:         yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
744:         ++yy_cp;
745:     }
746:     while ( yy_current_state != 200 );
747:     yy_cp = yy_last_accepting_cpos;
748:     yy_current_state = yy_last_accepting_state;
749:
750: yy_find_action:
751:     yy_act = yy_accept[yy_current_state];
752:
753:     YY_DO_BEFORE_ACTION;
754:
755:
756: do_action: /* This label is used only to access EOF actions. */
757:
758:
759:     switch ( yy_act )
760:     { /* beginning of action switch */
761:         case 0: /* must back up */
762:             /* undo the effects of YY_DO_BEFORE_ACTION */
763:             *yy_cp = yy_hold_char;
764:             yy_cp = yy_last_accepting_cpos;
765:             yy_current_state = yy_last_accepting_state;
766:             goto yy_find_action;
767:
768: case 1:
769:     YY_RULE_SETUP
770:     #line 20 "VarInfoParser.l"
771:     {return EQUAL_SY;}
772:     YY_BREAK
773: case 2:
774:     YY_RULE_SETUP
775:     #line 21 "VarInfoParser.l"
776:     {return CLOSE_SY;}
777:     YY_BREAK

```

778: case 3:
779: YY_RULE_SETUP
780: #line 22 "VarInfoParser.l"
781: {return SLASHCLOSE_SY;}
782: YY_BREAK
783: case 4:
784: YY_RULE_SETUP
785: #line 24 "VarInfoParser.l"
786: {return OPEN_VAR_SY;}
787: YY_BREAK
788: case 5:
789: YY_RULE_SETUP
790: #line 25 "VarInfoParser.l"
791: {return CLOSE_VAR_SY;}
792: YY_BREAK
793: case 6:
794: YY_RULE_SETUP
795: #line 27 "VarInfoParser.l"
796: {return OPEN_DRANGE_SY;}
797: YY_BREAK
798: case 7:
799: YY_RULE_SETUP
800: #line 28 "VarInfoParser.l"
801: {return CLOSE_DRANGE_SY;}
802: YY_BREAK
803: case 8:
804: YY_RULE_SETUP
805: #line 29 "VarInfoParser.l"
806: {return OPEN_OPTION_SY;}
807: YY_BREAK
808: case 9:
809: YY_RULE_SETUP
810: #line 30 "VarInfoParser.l"
811: {return CLOSE_OPTION_SY;}
812: YY_BREAK
813: case 10:
814: YY_RULE_SETUP
815: #line 31 "VarInfoParser.l"
816: {return OPEN_QUALIFIER_SY;}
817: YY_BREAK
818: case 11:

819: YY_RULE_SETUP
820: #line 32 "VarInfoParser.l"
821: {return CLOSE_QUALIFIER_SY;}
822: YY_BREAK
823: case 12:
824: YY_RULE_SETUP
825: #line 34 "VarInfoParser.l"
826: {return NAME_SY;}
827: YY_BREAK
828: case 13:
829: YY_RULE_SETUP
830: #line 35 "VarInfoParser.l"
831: {return KIND_SY;}
832: YY_BREAK
833: case 14:
834: YY_RULE_SETUP
835: #line 36 "VarInfoParser.l"
836: {return ID_SY;}
837: YY_BREAK
838: case 15:
839: YY_RULE_SETUP
840: #line 37 "VarInfoParser.l"
841: {return QUALIFIER_SY;}
842: YY_BREAK
843: case 16:
844: YY_RULE_SETUP
845: #line 38 "VarInfoParser.l"
846: {return DESCRIPTION_SY;}
847: YY_BREAK
848: case 17:
849: YY_RULE_SETUP
850: #line 39 "VarInfoParser.l"
851: {return PRECISION_SY;}
852: YY_BREAK
853: case 18:
854: YY_RULE_SETUP
855: #line 40 "VarInfoParser.l"
856: {return RANGE_MAX_SY;}
857: YY_BREAK
858: case 19:
859: YY_RULE_SETUP


```
860: #line 41 "VarInfoParser.l"
861: {return FORMAT_SY;}
862:   YY_BREAK
863: case 20:
864: YY_RULE_SETUP
865: #line 42 "VarInfoParser.l"
866: {return ACCURACY_SY;}
867:   YY_BREAK
868: case 21:
869: YY_RULE_SETUP
870: #line 43 "VarInfoParser.l"
871: {return RANGE_MIN_SY;}
872:   YY_BREAK
873: case 22:
874: YY_RULE_SETUP
875: #line 44 "VarInfoParser.l"
876: {return SCALE_FACTOR_SY;}
877:   YY_BREAK
878: case 23:
879: YY_RULE_SETUP
880: #line 45 "VarInfoParser.l"
881: {return UNITS_SY;}
882:   YY_BREAK
883: case 24:
884: YY_RULE_SETUP
885: #line 46 "VarInfoParser.l"
886: {return DEFAULT_VALUE_SY;}
887:   YY_BREAK
888: case 25:
889: YY_RULE_SETUP
890: #line 47 "VarInfoParser.l"
891: {return SCALE_UNITS_SY;}
892:   YY_BREAK
893: case 26:
894: YY_RULE_SETUP
895: #line 48 "VarInfoParser.l"
896: {return LENGTH_SY;}
897:   YY_BREAK
898: case 27:
899: YY_RULE_SETUP
900: #line 49 "VarInfoParser.l"
```

```

901: {return VALUE_SY;}
902:   YY_BREAK
903: case 28:
904: YY_RULE_SETUP
905: #line 51 "VarInfoParser.l"
906: {VarInfoParserlval.str = strdup(yytext+1, strlen(yytext)-2);return STRING;}
907:   YY_BREAK
908: case 29:
909: YY_RULE_SETUP
910: #line 52 "VarInfoParser.l"
911: {VarInfoParserlval.real = atof(yytext); return FLOAT;}
912:   YY_BREAK
913: case 30:
914: YY_RULE_SETUP
915: #line 53 "VarInfoParser.l"
916: {VarInfoParserlval.integer = atoi(yytext); return INT;}
917:   YY_BREAK
918: case 31:
919: YY_RULE_SETUP
920: #line 55 "VarInfoParser.l"
921: {/*ignore xteds comments*/}
922:   YY_BREAK
923: case 32:
924: YY_RULE_SETUP
925: #line 57 "VarInfoParser.l"
926: {/*ignore whitespace*/}
927:   YY_BREAK
928: case 33:
929: YY_RULE_SETUP
930: #line 58 "VarInfoParser.l"
931: {printf ("Invalid token  \"%s \"\n",yytext);}
932:   YY_BREAK
933: case 34:
934: YY_RULE_SETUP
935: #line 60 "VarInfoParser.l"
936: ECHO;
937:   YY_BREAK
938: #line 937 "lex.VarInfoParser.c"
939: case YY_STATE_EOF(INITIAL):
940:   yyterminate();
941:

```

```

942: case YY_END_OF_BUFFER:
943:     {
944:         /* Amount of text matched not including the EOB char. */
945:         int yy_amount_of_matched_text = (int) (yy_cp - yytext_ptr) - 1;
946:
947:         /* Undo the effects of YY_DO_BEFORE_ACTION. */
948:         *yy_cp = yy_hold_char;
949:         YY_RESTORE_YY_MORE_OFFSET
950:
951:         if ( yy_current_buffer->yy_buffer_status == YY_BUFFER_NEW )
952:             {
953:                 /* We're scanning a new file or input source. It's
954:                  * possible that this happened because the user
955:                  * just pointed yyin at a new source and called
956:                  * yylex(). If so, then we have to assure
957:                  * consistency between yy_current_buffer and our
958:                  * globals. Here is the right place to do so, because
959:                  * this is the first action (other than possibly a
960:                  * back-up) that will match for the new input source.
961:                  */
962:                 yy_n_chars = yy_current_buffer->yy_n_chars;
963:                 yy_current_buffer->yy_input_file = yyin;
964:                 yy_current_buffer->yy_buffer_status = YY_BUFFER_NORMAL;
965:             }
966:
967:         /* Note that here we test for yy_c_buf_p "<=" to the position
968:          * of the first EOB in the buffer, since yy_c_buf_p will
969:          * already have been incremented past the NUL character
970:          * (since all states make transitions on EOB to the
971:          * end-of-buffer state). Contrast this with the test
972:          * in input().
973:          */
974:         if ( yy_c_buf_p <= &yy_current_buffer->yy_ch_buf[yy_n_chars] )
975:             { /* This was really a NUL. */
976:                 yy_state_type yy_next_state;
977:
978:                 yy_c_buf_p = yytext_ptr + yy_amount_of_matched_text;
979:
980:                 yy_current_state = yy_get_previous_state();
981:
982:                 /* Okay, we're now positioned to make the NUL

```

```

983:      * transition. We couldn't have
984:      * yy_get_previous_state() go ahead and do it
985:      * for us because it doesn't know how to deal
986:      * with the possibility of jamming (and we don't
987:      * want to build jamming into it because then it
988:      * will run more slowly).
989:      */
990:
991:      yy_next_state = yy_try_NUL_trans( yy_current_state );
992:
993:      yy_bp = yytext_ptr + YY_MORE_ADJ;
994:
995:      if ( yy_next_state )
996:      {
997:          /* Consume the NUL. */
998:          yy_cp = ++yy_c_buf_p;
999:          yy_current_state = yy_next_state;
1000:          goto yy_match;
1001:      }
1002:
1003:      else
1004:      {
1005:          yy_cp = yy_last_accepting_cpos;
1006:          yy_current_state = yy_last_accepting_state;
1007:          goto yy_find_action;
1008:      }
1009:  }
1010:
1011:  else switch ( yy_get_next_buffer() )
1012:  {
1013:      case EOB_ACT_END_OF_FILE:
1014:      {
1015:          yy_did_buffer_switch_on_eof = 0;
1016:
1017:          if ( yywrap() )
1018:          {
1019:              /* Note: because we've taken care in
1020:               * yy_get_next_buffer() to have set up
1021:               * yytext, we can now set up
1022:               * yy_c_buf_p so that if some total
1023:               * hoser (like flex itself) wants to

```

```

1024:         * call the scanner after we return the
1025:         * YY_NULL, it'll still work - another
1026:         * YY_NULL will get returned.
1027:         */
1028:         yy_c_buf_p = yytext_ptr + YY_MORE_ADJ;
1029:
1030:         yy_act = YY_STATE_EOF(YY_START);
1031:         goto do_action;
1032:     }
1033:
1034:     else
1035:     {
1036:         if ( ! yy.did_buffer_switch_on_eof )
1037:             YY_NEW_FILE;
1038:     }
1039:     break;
1040: }
1041:
1042: case EOB_ACT_CONTINUE_SCAN:
1043:     yy_c_buf_p =
1044:         yytext_ptr + yy_amount_of_matched_text;
1045:
1046:     yy_current_state = yy_get_previous_state();
1047:
1048:     yy_cp = yy_c_buf_p;
1049:     yy_bp = yytext_ptr + YY_MORE_ADJ;
1050:     goto yy_match;
1051:
1052: case EOB_ACT_LAST_MATCH:
1053:     yy_c_buf_p =
1054:         &yy_current_buffer->yy_ch_buf[yy_n_chars];
1055:
1056:     yy_current_state = yy_get_previous_state();
1057:
1058:     yy_cp = yy_c_buf_p;
1059:     yy_bp = yytext_ptr + YY_MORE_ADJ;
1060:     goto yy_find_action;
1061: }
1062: break;
1063: }
1064:

```

```

1065: default:
1066:     YY_FATAL_ERROR(
1067:         "fatal flex scanner internal error--no action found" );
1068: } /* end of action switch */
1069:     } /* end of scanning one token */
1070: } /* end of yylex */
1071:
1072:
1073: /* yy_get_next_buffer - try to read in a new buffer
1074:  *
1075:  * Returns a code representing an action:
1076:  *   EOB_ACT_LAST_MATCH -
1077:  *   EOB_ACT_CONTINUE_SCAN - continue scanning from current position
1078:  *   EOB_ACT_END_OF_FILE - end of file
1079:  */
1080:
1081: static int yy_get_next_buffer()
1082: {
1083:     register char *dest = yy_current_buffer->yy_ch_buf;
1084:     register char *source = yytext_ptr;
1085:     register int number_to_move, i;
1086:     int ret_val;
1087:
1088:     if ( yy_c_buf_p > &yy_current_buffer->yy_ch_buf[yy_n_chars + 1] )
1089:         YY_FATAL_ERROR(
1090:             "fatal flex scanner internal error--end of buffer missed" );
1091:
1092:     if ( yy_current_buffer->yy_fill_buffer == 0 )
1093:         { /* Don't try to fill the buffer, so this is an EOF. */
1094:             if ( yy_c_buf_p - yytext_ptr - YY_MORE_ADJ == 1 )
1095:                 {
1096:                     /* We matched a single character, the EOB, so
1097:                      * treat this as a final EOF.
1098:                      */
1099:                     return EOB_ACT_END_OF_FILE;
1100:                 }
1101:
1102:             else
1103:                 {
1104:                     /* We matched some text prior to the EOB, first
1105:                      * process it.

```

```

1106:         */
1107:         return EOB_ACT_LAST_MATCH;
1108:     }
1109: }
1110:
1111: /* Try to read more data. */
1112:
1113: /* First move last chars to start of buffer. */
1114: number_to_move = (int) (yy_c_buf_p - yytext_ptr) - 1;
1115:
1116: for ( i = 0; i < number_to_move; ++i )
1117:     *(dest++) = *(source++);
1118:
1119: if ( yy_current_buffer->yy_buffer_status == YY_BUFFER_EOF_PENDING )
1120:     /* don't do the read, it's not guaranteed to return an EOF,
1121:      * just force an EOF
1122:      */
1123:     yy_current_buffer->yy_n_chars = yy_n_chars = 0;
1124:
1125: else
1126:     {
1127:         int num_to_read =
1128:             yy_current_buffer->yy_buf_size - number_to_move - 1;
1129:
1130:         while ( num_to_read <= 0 )
1131:             { /* Not enough room in the buffer - grow it. */
1132: #ifdef YY_USES_REJECT
1133:                 YY_FATAL_ERROR(
1134: "input buffer overflow, can't enlarge buffer because scanner uses REJECT" );
1135: #else
1136:
1137:                 /* just a shorter name for the current buffer */
1138:                 YY_BUFFER_STATE b = yy_current_buffer;
1139:
1140:                 int yy_c_buf_p_offset =
1141:                     (int) (yy_c_buf_p - b->yy_ch_buf);
1142:
1143:                 if ( b->yy_is_our_buffer )
1144:                     {
1145:                         int new_size = b->yy_buf_size * 2;
1146:

```

```

1147:         if ( new_size <= 0 )
1148:             b->yy_buf_size += b->yy_buf_size / 8;
1149:         else
1150:             b->yy_buf_size *= 2;
1151:
1152:         b->yy_ch_buf = (char *)
1153:             /* Include room in for 2 EOB chars. */
1154:             yy_flex_realloc( (void *) b->yy_ch_buf,
1155:                 b->yy_buf_size + 2 );
1156:     }
1157:     else
1158:         /* Can't grow it, we don't own it. */
1159:         b->yy_ch_buf = 0;
1160:
1161:     if ( ! b->yy_ch_buf )
1162:         YY_FATAL_ERROR(
1163:             "fatal error - scanner input buffer overflow" );
1164:
1165:     yy_c_buf_p = &b->yy_ch_buf[yy_c_buf_p_offset];
1166:
1167:     num_to_read = yy_current_buffer->yy_buf_size -
1168:         number_to_move - 1;
1169: #endif
1170:     }
1171:
1172:     if ( num_to_read > YY_READ_BUF_SIZE )
1173:         num_to_read = YY_READ_BUF_SIZE;
1174:
1175:     /* Read in more data. */
1176:     YY_INPUT( (&yy_current_buffer->yy_ch_buf[number_to_move]),
1177:         yy_n_chars, num_to_read );
1178:
1179:     yy_current_buffer->yy_n_chars = yy_n_chars;
1180: }
1181:
1182: if ( yy_n_chars == 0 )
1183: {
1184:     if ( number_to_move == YY_MORE_ADJ )
1185:     {
1186:         ret_val = EOB_ACT_END_OF_FILE;
1187:         yyrestart( yyin );

```



```

1188:     }
1189:
1190:     else
1191:     {
1192:         ret_val = EOB_ACT_LAST_MATCH;
1193:         yy_current_buffer->yy_buffer_status =
1194:             YY_BUFFER_EOF_PENDING;
1195:     }
1196: }
1197:
1198: else
1199:     ret_val = EOB_ACT_CONTINUE_SCAN;
1200:
1201: yy_n_chars += number_to_move;
1202: yy_current_buffer->yy_ch_buf[yy_n_chars] = YY_END_OF_BUFFER_CHAR;
1203: yy_current_buffer->yy_ch_buf[yy_n_chars + 1] = YY_END_OF_BUFFER_CHAR;
1204:
1205: yytext_ptr = &yy_current_buffer->yy_ch_buf[0];
1206:
1207: return ret_val;
1208: }
1209:
1210:
1211: /* yy_get_previous_state - get the state just before the EOB char was reached */
1212:
1213: static yy_state_type yy_get_previous_state()
1214: {
1215:     register yy_state_type yy_current_state;
1216:     register char *yy_cp;
1217:
1218:     yy_current_state = yy_start;
1219:
1220:     for ( yy_cp = yytext_ptr + YY_MORE_ADJ; yy_cp < yy_c_buf_p; ++yy_cp )
1221:     {
1222:         register YY_CHAR yy_c = (*yy_cp ? yy_ec[YY_SC_TO_UI(*yy_cp)] : 1);
1223:         if ( yy_accept[yy_current_state] )
1224:         {
1225:             yy_last_accepting_state = yy_current_state;
1226:             yy_last_accepting_cpos = yy_cp;
1227:         }
1228:         while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )

```

```

1229:     {
1230:         yy_current_state = (int) yy_def[yy_current_state];
1231:         if ( yy_current_state >= 201 )
1232:             yy_c = yy_meta[(unsigned int) yy_c];
1233:     }
1234:     yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
1235: }
1236:
1237: return yy_current_state;
1238: }
1239:
1240:
1241: /* yy_try_NUL_trans - try to make a transition on the NUL character
1242:  *
1243:  * synopsis
1244:  *     next_state = yy_try_NUL_trans( current_state );
1245:  */
1246:
1247: #ifdef YY_USE_PROTOS
1248: static yy_state_type yy_try_NUL_trans( yy_state_type yy_current_state )
1249: #else
1250: static yy_state_type yy_try_NUL_trans( yy_current_state )
1251: yy_state_type yy_current_state;
1252: #endif
1253: {
1254:     register int yy_is_jam;
1255:     register char *yy_cp = yy_c_buf_p;
1256:
1257:     register YY_CHAR yy_c = 1;
1258:     if ( yy_accept[yy_current_state] )
1259:     {
1260:         yy_last_accepting_state = yy_current_state;
1261:         yy_last_accepting_cpos = yy_cp;
1262:     }
1263:     while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
1264:     {
1265:         yy_current_state = (int) yy_def[yy_current_state];
1266:         if ( yy_current_state >= 201 )
1267:             yy_c = yy_meta[(unsigned int) yy_c];
1268:     }
1269:     yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];

```

```

1270: yy_is_jam = (yy_current_state == 200);
1271:
1272: return yy_is_jam ? 0 : yy_current_state;
1273: }
1274:
1275:
1276: #ifndef YY_NO_UNPUT
1277: #ifdef YY_USE_PROTOS
1278: static void yyunput( int c, register char *yy_bp )
1279: #else
1280: static void yyunput( c, yy_bp )
1281: int c;
1282: register char *yy_bp;
1283: #endif
1284: {
1285: register char *yy_cp = yy_c_buf_p;
1286:
1287: /* undo effects of setting up yytext */
1288: *yy_cp = yy_hold_char;
1289:
1290: if ( yy_cp < yy_current_buffer->yy_ch_buf + 2 )
1291:     { /* need to shift things up to make room */
1292:         /* +2 for EOB chars. */
1293:         register int number_to_move = yy_n_chars + 2;
1294:         register char *dest = &yy_current_buffer->yy_ch_buf[
1295:             yy_current_buffer->yy_buf_size + 2];
1296:         register char *source =
1297:             &yy_current_buffer->yy_ch_buf[number_to_move];
1298:
1299:         while ( source > yy_current_buffer->yy_ch_buf )
1300:             *--dest = *--source;
1301:
1302:         yy_cp += (int) (dest - source);
1303:         yy_bp += (int) (dest - source);
1304:         yy_current_buffer->yy_n_chars =
1305:             yy_n_chars = yy_current_buffer->yy_buf_size;
1306:
1307:         if ( yy_cp < yy_current_buffer->yy_ch_buf + 2 )
1308:             YY_FATAL_ERROR( "flex scanner push-back overflow" );
1309:     }
1310:

```

```

1311:  *--yy_cp = (char) c;
1312:
1313:
1314:  yytext_ptr = yy_bp;
1315:  yy_hold_char = *yy_cp;
1316:  yy_c_buf_p = yy_cp;
1317:  }
1318: #endif      /* ifndef YY_NO_UNPUT */
1319:
1320:
1321: #ifndef YY_NO_INPUT
1322: #ifdef __cplusplus
1323: static int yyinput()
1324: #else
1325: static int input()
1326: #endif
1327: {
1328:   int c;
1329:
1330:   *yy_c_buf_p = yy_hold_char;
1331:
1332:   if ( *yy_c_buf_p == YY_END_OF_BUFFER_CHAR )
1333:     {
1334:       /* yy_c_buf_p now points to the character we want to return.
1335:        * If this occurs *before* the EOB characters, then it's a
1336:        * valid NUL; if not, then we've hit the end of the buffer.
1337:        */
1338:       if ( yy_c_buf_p < &yy_current_buffer->yy_ch_buf[yy_n_chars] )
1339:         /* This was really a NUL. */
1340:         *yy_c_buf_p = '\0';
1341:
1342:       else
1343:         { /* need more input */
1344:           int offset = yy_c_buf_p - yytext_ptr;
1345:           ++yy_c_buf_p;
1346:
1347:           switch ( yy_get_next_buffer() )
1348:             {
1349:               case EOB_ACT_LAST_MATCH:
1350:                 /* This happens because yy_g_n_b()
1351:                  * sees that we've accumulated a

```

```

1352:         * token and flags that we need to
1353:         * try matching the token before
1354:         * proceeding. But for input(),
1355:         * there's no matching to consider.
1356:         * So convert the EOB_ACT_LAST_MATCH
1357:         * to EOB_ACT_END_OF_FILE.
1358:         */
1359:
1360:         /* Reset buffer status. */
1361:         yyrestart( yyin );
1362:
1363:         /* fall through */
1364:
1365:         case EOB_ACT_END_OF_FILE:
1366:         {
1367:             if ( yywrap() )
1368:                 return EOF;
1369:
1370:             if ( ! yy_did_buffer_switch_on_eof )
1371:                 YY_NEW_FILE;
1372: #ifdef __cplusplus
1373:             return yyinput();
1374: #else
1375:             return input();
1376: #endif
1377:         }
1378:
1379:         case EOB_ACT_CONTINUE_SCAN:
1380:             yy_c_buf_p = yytext_ptr + offset;
1381:             break;
1382:         }
1383:     }
1384: }
1385:
1386: c = *(unsigned char *) yy_c_buf_p; /* cast for 8-bit char's */
1387: *yy_c_buf_p = '\0'; /* preserve yytext */
1388: yy_hold_char = *++yy_c_buf_p;
1389:
1390:
1391: return c;
1392: }

```

```

1393: #endif /* YY_NO_INPUT */
1394:
1395: #ifdef YY_USE_PROTOS
1396: void yyrestart( FILE *input_file )
1397: #else
1398: void yyrestart( input_file )
1399: FILE *input_file;
1400: #endif
1401: {
1402:   if ( ! yy_current_buffer )
1403:     yy_current_buffer = yy_create_buffer( yyin, YY_BUF_SIZE );
1404:
1405:   yy_init_buffer( yy_current_buffer, input_file );
1406:   yy_load_buffer_state();
1407: }
1408:
1409:
1410: #ifdef YY_USE_PROTOS
1411: void yy_switch_to_buffer( YY_BUFFER_STATE new_buffer )
1412: #else
1413: void yy_switch_to_buffer( new_buffer )
1414: YY_BUFFER_STATE new_buffer;
1415: #endif
1416: {
1417:   if ( yy_current_buffer == new_buffer )
1418:     return;
1419:
1420:   if ( yy_current_buffer )
1421:     {
1422:       /* Flush out information for old buffer. */
1423:       *yy_c_buf_p = yy_hold_char;
1424:       yy_current_buffer->yy_buf_pos = yy_c_buf_p;
1425:       yy_current_buffer->yy_n_chars = yy_n_chars;
1426:     }
1427:
1428:   yy_current_buffer = new_buffer;
1429:   yy_load_buffer_state();
1430:
1431:   /* We don't actually know whether we did this switch during
1432:    * EOF (yywrap()) processing, but the only time this flag
1433:    * is looked at is after yywrap() is called, so it's safe

```

```

1434:  * to go ahead and always set it.
1435:  */
1436:  yy_did_buffer_switch_on_eof = 1;
1437:  }
1438:
1439:
1440: #ifdef YY_USE_PROTOS
1441: void yy_load_buffer_state( void )
1442: #else
1443: void yy_load_buffer_state()
1444: #endif
1445: {
1446:  yy_n_chars = yy_current_buffer->yy_n_chars;
1447:  yytext_ptr = yy_c_buf_p = yy_current_buffer->yy_buf_pos;
1448:  yyin = yy_current_buffer->yy_input_file;
1449:  yy_hold_char = *yy_c_buf_p;
1450:  }
1451:
1452:
1453: #ifdef YY_USE_PROTOS
1454: YY_BUFFER_STATE yy_create_buffer( FILE *file, int size )
1455: #else
1456: YY_BUFFER_STATE yy_create_buffer( file, size )
1457: FILE *file;
1458: int size;
1459: #endif
1460: {
1461:  YY_BUFFER_STATE b;
1462:
1463:  b = (YY_BUFFER_STATE) yy_flex_alloc( sizeof( struct yy_buffer_state ) );
1464:  if ( ! b )
1465:      YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );
1466:
1467:  b->yy_buf_size = size;
1468:
1469:  /* yy_ch_buf has to be 2 characters longer than the size given because
1470:   * we need to put in 2 end-of-buffer characters.
1471:   */
1472:  b->yy_ch_buf = (char *) yy_flex_alloc( b->yy_buf_size + 2 );
1473:  if ( ! b->yy_ch_buf )
1474:      YY_FATAL_ERROR( "out of dynamic memory in yy_create_buffer()" );

```

```

1475:
1476:  b->yy_is_our_buffer = 1;
1477:
1478:  yy_init_buffer( b, file );
1479:
1480:  return b;
1481:  }
1482:
1483:
1484: #ifdef YY_USE_PROTOS
1485: void yy_delete_buffer( YY_BUFFER_STATE b )
1486: #else
1487: void yy_delete_buffer( b )
1488: YY_BUFFER_STATE b;
1489: #endif
1490:  {
1491:  if ( ! b )
1492:      return;
1493:
1494:  if ( b == yy_current_buffer )
1495:      yy_current_buffer = (YY_BUFFER_STATE) 0;
1496:
1497:  if ( b->yy_is_our_buffer )
1498:      yy_flex_free( (void *) b->yy_ch_buf );
1499:
1500:  yy_flex_free( (void *) b );
1501:  }
1502:
1503:
1504:
1505: #ifdef YY_USE_PROTOS
1506: void yy_init_buffer( YY_BUFFER_STATE b, FILE *file )
1507: #else
1508: void yy_init_buffer( b, file )
1509: YY_BUFFER_STATE b;
1510: FILE *file;
1511: #endif
1512:
1513:
1514:  {
1515:  yy_flush_buffer( b );

```



```

1516:
1517:  b->yy_input_file = file;
1518:  b->yy_fill_buffer = 1;
1519:
1520: #if YY_ALWAYS_INTERACTIVE
1521:  b->yy_is_interactive = 1;
1522: #else
1523: #if YY_NEVER_INTERACTIVE
1524:  b->yy_is_interactive = 0;
1525: #else
1526:  b->yy_is_interactive = file ? (isatty( fileno(file) ) > 0) : 0;
1527: #endif
1528: #endif
1529: }
1530:
1531:
1532: #ifdef YY_USE_PROTOS
1533: void yy_flush_buffer( YY_BUFFER_STATE b )
1534: #else
1535: void yy_flush_buffer( b )
1536: YY_BUFFER_STATE b;
1537: #endif
1538:
1539: {
1540:  if ( ! b )
1541:      return;
1542:
1543:  b->yy_n_chars = 0;
1544:
1545:  /* We always need two end-of-buffer characters.  The first causes
1546:   * a transition to the end-of-buffer state.  The second causes
1547:   * a jam in that state.
1548:   */
1549:  b->yy_ch_buf[0] = YY_END_OF_BUFFER_CHAR;
1550:  b->yy_ch_buf[1] = YY_END_OF_BUFFER_CHAR;
1551:
1552:  b->yy_buf_pos = &b->yy_ch_buf[0];
1553:
1554:  b->yy_at_bol = 1;
1555:  b->yy_buffer_status = YY_BUFFER_NEW;
1556:

```

```

1557:  if ( b == yy_current_buffer )
1558:      yy_load_buffer_state();
1559:  }
1560:
1561:
1562: #ifndef YY_NO_SCAN_BUFFER
1563: #ifdef YY_USE_PROTOS
1564: YY_BUFFER_STATE yy_scan_buffer( char *base, yy_size_t size )
1565: #else
1566: YY_BUFFER_STATE yy_scan_buffer( base, size )
1567: char *base;
1568: yy_size_t size;
1569: #endif
1570: {
1571:  YY_BUFFER_STATE b;
1572:
1573:  if ( size < 2 ||
1574:      base[size-2] != YY_END_OF_BUFFER_CHAR ||
1575:      base[size-1] != YY_END_OF_BUFFER_CHAR )
1576:      /* They forgot to leave room for the EOB's. */
1577:      return 0;
1578:
1579:  b = (YY_BUFFER_STATE) yy_flex_alloc( sizeof( struct yy_buffer_state ) );
1580:  if ( ! b )
1581:      YY_FATAL_ERROR( "out of dynamic memory in yy_scan_buffer()" );
1582:
1583:  b->yy_buf_size = size - 2;  /* "- 2" to take care of EOB's */
1584:  b->yy_buf_pos = b->yy_ch_buf = base;
1585:  b->yy_is_our_buffer = 0;
1586:  b->yy_input_file = 0;
1587:  b->yy_n_chars = b->yy_buf_size;
1588:  b->yy_is_interactive = 0;
1589:  b->yy_at_bol = 1;
1590:  b->yy_fill_buffer = 0;
1591:  b->yy_buffer_status = YY_BUFFER_NEW;
1592:
1593:  yy_switch_to_buffer( b );
1594:
1595:  return b;
1596: }
1597: #endif

```

```

1598:
1599:
1600: #ifndef YY_NO_SCAN_STRING
1601: #ifdef YY_USE_PROTOS
1602: YY_BUFFER_STATE yy_scan_string( yyconst char *yy_str )
1603: #else
1604: YY_BUFFER_STATE yy_scan_string( yy_str )
1605: yyconst char *yy_str;
1606: #endif
1607: {
1608:   int len;
1609:   for ( len = 0; yy_str[len]; ++len )
1610:     ;
1611:
1612:   return yy_scan_bytes( yy_str, len );
1613: }
1614: #endif
1615:
1616:
1617: #ifndef YY_NO_SCAN_BYTES
1618: #ifdef YY_USE_PROTOS
1619: YY_BUFFER_STATE yy_scan_bytes( yyconst char *bytes, int len )
1620: #else
1621: YY_BUFFER_STATE yy_scan_bytes( bytes, len )
1622: yyconst char *bytes;
1623: int len;
1624: #endif
1625: {
1626:   YY_BUFFER_STATE b;
1627:   char *buf;
1628:   yy_size_t n;
1629:   int i;
1630:
1631:   /* Get memory for full buffer, including space for trailing EOB's. */
1632:   n = len + 2;
1633:   buf = (char *) yy_flex_alloc( n );
1634:   if ( ! buf )
1635:     YY_FATAL_ERROR( "out of dynamic memory in yy_scan_bytes()" );
1636:
1637:   for ( i = 0; i < len; ++i )
1638:     buf[i] = bytes[i];

```

```

1639:
1640:  buf[len] = buf[len+1] = YY_END_OF_BUFFER_CHAR;
1641:
1642:  b = yy_scan_buffer( buf, n );
1643:  if ( ! b )
1644:      YY_FATAL_ERROR( "bad buffer in yy_scan_bytes()" );
1645:
1646:  /* It's okay to grow etc. this buffer, and we should throw it
1647:   * away when we're done.
1648:   */
1649:  b->yy_is_our_buffer = 1;
1650:
1651:  return b;
1652:  }
1653: #endif
1654:
1655:
1656: #ifndef YY_NO_PUSH_STATE
1657: #ifdef YY_USE_PROTOS
1658: static void yy_push_state( int new_state )
1659: #else
1660: static void yy_push_state( new_state )
1661: int new_state;
1662: #endif
1663: {
1664:  if ( yy_start_stack_ptr >= yy_start_stack_depth )
1665:      {
1666:          yy_size_t new_size;
1667:
1668:          yy_start_stack_depth += YY_START_STACK_INCR;
1669:          new_size = yy_start_stack_depth * sizeof( int );
1670:
1671:          if ( ! yy_start_stack )
1672:              yy_start_stack = (int *) yy_flex_alloc( new_size );
1673:
1674:          else
1675:              yy_start_stack = (int *) yy_flex_realloc(
1676:                  (void *) yy_start_stack, new_size );
1677:
1678:          if ( ! yy_start_stack )
1679:              YY_FATAL_ERROR(

```

```

1680:         "out of memory expanding start-condition stack" );
1681:     }
1682:
1683: yy_start_stack[yy_start_stack_ptr++] = YY_START;
1684:
1685: BEGIN(new_state);
1686: }
1687: #endif
1688:
1689:
1690: #ifndef YY_NO_POP_STATE
1691: static void yy_pop_state()
1692: {
1693:     if ( --yy_start_stack_ptr < 0 )
1694:         YY_FATAL_ERROR( "start-condition stack underflow" );
1695:
1696: BEGIN(yy_start_stack[yy_start_stack_ptr]);
1697: }
1698: #endif
1699:
1700:
1701: #ifndef YY_NO_TOP_STATE
1702: static int yy_top_state()
1703: {
1704:     return yy_start_stack[yy_start_stack_ptr - 1];
1705: }
1706: #endif
1707:
1708: #ifndef YY_EXIT_FAILURE
1709: #define YY_EXIT_FAILURE 2
1710: #endif
1711:
1712: #ifdef YY_USE_PROTOS
1713: static void yy_fatal_error( yyconst char msg[] )
1714: #else
1715: static void yy_fatal_error( msg )
1716: char msg[];
1717: #endif
1718: {
1719:     (void) fprintf( stderr, "%s \n", msg );
1720:     exit( YY_EXIT_FAILURE );

```

```

1721:  }
1722:
1723:
1724:
1725: /* Redefine yyless() so it works in section 3 code. */
1726:
1727: #undef yyless
1728: #define yyless(n) \
1729:  do \
1730:  { \
1731:    /* Undo effects of setting up yytext. */ \
1732:    yytext[yyleng] = yy_hold_char; \
1733:    yy_c_buf_p = yytext + n; \
1734:    yy_hold_char = *yy_c_buf_p; \
1735:    *yy_c_buf_p = '\0'; \
1736:    yyleng = n; \
1737:  } \
1738:  while ( 0 )
1739:
1740:
1741: /* Internal utility routines. */
1742:
1743: #ifndef yytext_ptr
1744: #ifdef YY_USE_PROTOS
1745: static void yy_flex_strncpy( char *s1, yyconst char *s2, int n )
1746: #else
1747: static void yy_flex_strncpy( s1, s2, n )
1748: char *s1;
1749: yyconst char *s2;
1750: int n;
1751: #endif
1752: {
1753:  register int i;
1754:  for ( i = 0; i < n; ++i )
1755:    s1[i] = s2[i];
1756: }
1757: #endif
1758:
1759: #ifdef YY_NEED_STRLEN
1760: #ifdef YY_USE_PROTOS
1761: static int yy_flex_strlen( yyconst char *s )

```

```

1762: #else
1763: static int yy_flex_strlen( s )
1764: yyconst char *s;
1765: #endif
1766: {
1767:     register int n;
1768:     for ( n = 0; s[n]; ++n )
1769:         ;
1770:
1771:     return n;
1772: }
1773: #endif
1774:
1775:
1776: #ifdef YY_USE_PROTOS
1777: static void *yy_flex_alloc( yy_size_t size )
1778: #else
1779: static void *yy_flex_alloc( size )
1780: yy_size_t size;
1781: #endif
1782: {
1783:     return (void *) malloc( size );
1784: }
1785:
1786: #ifdef YY_USE_PROTOS
1787: static void *yy_flex_realloc( void *ptr, yy_size_t size )
1788: #else
1789: static void *yy_flex_realloc( ptr, size )
1790: void *ptr;
1791: yy_size_t size;
1792: #endif
1793: {
1794:     /* The cast to (char *) in the following accommodates both
1795:      * implementations that use char* generic pointers, and those
1796:      * that use void* generic pointers. It works with the latter
1797:      * because both ANSI C and C++ allow castless assignment from
1798:      * any pointer type to void*, and deal with argument conversions
1799:      * as though doing an assignment.
1800:      */
1801:     return (void *) realloc( (char *) ptr, size );
1802: }

```

```

1803:
1804: #ifdef YY_USE_PROTOS
1805: static void yy_free( void *ptr )
1806: #else
1807: static void yy_free( ptr )
1808: void *ptr;
1809: #endif
1810: {
1811:   free( ptr );
1812: }
1813:
1814: #if YY_MAIN
1815: int main()
1816: {
1817:   yylex();
1818:   return 0;
1819: }
1820: #endif
1821: #line 60 "VarInfoParser.l"
1822:
1823:
1824: int yywrap() {return 1;}
1825: void VarInfoParsererror(char *s)
1826: {
1827:   printf("Error occurred at token %s (error is %s) \n",yytext, s);
1828: }

```


File: sdm/common/VarInfoParser/VarInfoParser.y

```
1: %{
2: #include <stdio.h>
3: #include <string.h>
4: #include "../xTEDS/xTEDSParser.h"
5: %}
6:
7:
8: %token EQUAL_SY CLOSE_SY SLASHCLOSE_SY OPEN_XML_SY CLOSE_xTEDS_SY
OPEN_xTEDS_SY OPEN_APP_SY
9: %token OPEN_VAR_SY CLOSE_VAR_SY OPEN_DRANGE_SY CLOSE_DRANGE_SY
OPEN_OPTION_SY OPEN_CURVE_SY
10: %token CLOSE_CURVE_SY OPEN_COEFF_SY OPEN_DATA_MSG_SY
CLOSE_DATA_MSG_SY OPEN_VARIABLE_REF_SY
11: %token OPEN_COMMAND_MSG_SY CLOSE_COMMAND_MSG_SY NAME_SY KIND_SY
ID_SY CLOSE_ORIENTATION_SY
12: %token QUALIFIER_SY DESCRIPTION_SY MANUFACTURER_ID_SY VERSION_SY
MODEL_ID_SY VERSION_LETTER_SY
13: %token SERIAL_NUMBER_SY CALIBRATION_DATE_SY SENSITIVITY_AT_REF_SY
REF_FREQ_SY REF_TEMP_SY
14: %token MEASUREMENT_RANGE_SY ELECTRICAL_OUTPUT_SY QUALITY_FACTOR_SY
TEMP_COEFF_SY DIRECTION_XYZ_SY
15: %token CAL_DUE_DATE_SY POWER_REQS_SY VALUE_SY ALARM_SY
MSG_ARRIVAL_SY MSG_RATE_SY
16: %token STRING FLOAT INT PRECISION_SY RANGE_MAX_SY CLOSE_LOCATION_SY
CLOSE_ORIENTATION_SY
17: %token FORMAT_SY ACCURACY_SY RANGE_MIN_SY SCALE_FACTOR_SY UNITS_SY
DEFAULT_VALUE_SY
18: %token OPEN_DEVICE_SY SCALE_UNITS_SY LENGTH_SY EXPONENT_SY
SCHEMA_LOCATION_SY XMLNS_SY XMLNS_XSI_SY
19: %token CLOSE_OPTION_SY OPEN_INTERFACE_SY OPEN_COMMAND_SY
OPEN_NOTIFICATION_SY OPEN_REQUEST_SY
20: %token OPEN_FAULT_MSG_SY COMPONENT_KEY_SY SPA_U_HUB_SY SPA_U_PORT_SY
EXTENDS_SY
21: %token CLOSE_COMMAND_SY CLOSE_NOTIFICATION_SY CLOSE_REQUEST_SY
CLOSE_FAULT_MSG_SY OPEN_QUALIFIER_SY
22: %token CLOSE_QUALIFIER_SY CLOSE_APP_SY CLOSE_DEVICE_SY
CLOSE_INTERFACE_SY MEMORY_MINIMUM_SY
23: %token OPERATING_SYSTEM_SY PATH_FOR_ASSEMBLY_SY
PATH_ON_SPACECRAFT_SY X_SY Y_SY Z_SY AXIS_SY ANGLE_SY
24: %token OPEN_LOCATION_SY OPEN_ORIENTATION_SY CLOSE_XML_SY ENCODING_SY
STANDALONE_SY CLOSE_VARIABLE_REF_SY
25: %token CLOSE_COEFF_SY
```

```

26:
27: %type<str> STRING
28: %type<integer> INT
29: %type<real> FLOAT
30: %type<var> VAR_ATTRIBUTES VAR_ATTRIBUTE VARIABLE VAR_HEAD
VAR_WITH_SUBELEMENTS VAR_NO_SUBELEMENTS VAR_SUBELEMENT
VAR_SUBELEMENTS
31: %type<var> VAR_ELEMENTS VAR_QUALIFIERS
32: %type<qual> QUALIFIERS_SECTION QUALIFIERS QUALIFIERS_WITH_SUBELEMENTS
QUALIFIERS_NO_SUBELEMENTS QUALIFIERS_HEAD
33: %type<qual> QUALIFIERS_ATTRIBUTE QUALIFIERS_ATTRIBUTES
34: %type<coef> CURVE_COEFFS CURVE_COEFF COEFF_ATTRIBUTES COEFF_ATTRIBUTE
35: %type<curve> CURVE_HEAD CURVE_ATTRIBUTES CURVE_ATTRIBUTE CURVE
36: %type<curveoption> DRANGE_OPTIONS DRANGE_OPTION OPTION_ATTRIBUTES
OPTION_ATTRIBUTE
37: %type<drange> DRANGE_HEAD DRANGE_ATTRIBUTES DRANGE_ATTRIBUTE DRANGE
38:
39: %union
40: {
41: int integer;
42: float real;
43: char* str;
44: struct variable_data* var;
45: struct qualifier_data* qual;
46: struct coefficient_data* coef;
47: struct curve_data* curve;
48: struct option_data* curveoption;
49: struct drange_data* drange;
50: }
51:
52:
53: %%
54:
55: VARIABLE : VAR_WITH_SUBELEMENTS
56: {
57: $$=$1
58: }
59: | VAR_NO_SUBELEMENTS
60: {
61: $$=$1
62: }
63: ;

```

```

64:
65: VAR_WITH_SUBELEMENTS :   VAR_HEAD           CLOSE_SY           VAR_ELEMENTS
CLOSE_VAR_SY CLOSE_SY
66: {
67:     $$ = merge_variables($1,$3);
68: }
69:     ;
70:
71: VAR_NO_SUBELEMENTS    :   VAR_HEAD SLASHCLOSE_SY
72:     ;
73:
74: VAR_HEAD              :   OPEN_VAR_SY VAR_ATTRIBUTES
75: {
76:     $$= $2
77: }
78:     ;
79:
80: VAR_ATTRIBUTES        :   VAR_ATTRIBUTES VAR_ATTRIBUTE
81: {
82:     $$= merge_variables($1,$2);
83: }
84:     |   /*empty*/
85: {
86:     $$=NULL;
87: }
88:     ;
89:
90: VAR_ATTRIBUTE         :   NAME_SY EQUAL_SY STRING
91: {
92:     variable* temp;
93:     temp = new_variable();
94:     temp->name = $3;
95:     $$ = temp;
96: }
97:     |   KIND_SY EQUAL_SY STRING
98: {
99:     variable* temp;
100:     temp = new_variable();
101:     temp->kind = $3;
102:     $$ = temp;
103: }

```

```

104:      |  FORMAT_SY EQUAL_SY STRING
105: {
106:     variable* temp;
107:     temp = new_variable();
108:     temp->format = $3;
109:     $$ = temp;
110: }
111:      |  QUALIFIER_SY EQUAL_SY STRING
112: {
113:     variable* temp;
114:     temp = new_variable();
115:     temp->qualifier = $3;
116:     printf("Qualifier field has been deprecated! \n");
117:     $$ = temp;
118: }
119:      |  ID_SY EQUAL_SY STRING
120: {
121:     variable* temp;
122:     temp = new_variable();
123:     temp->id = $3;
124:     $$ = temp;
125: }
126:      |  DESCRIPTION_SY EQUAL_SY STRING
127: {
128:     variable* temp;
129:     temp = new_variable();
130:     temp->description = $3;
131:     $$ = temp;
132: }
133:      |  RANGE_MIN_SY EQUAL_SY STRING
134: {
135:     variable* temp;
136:     temp = new_variable();
137:     temp->range_min = $3;
138:     $$ = temp;
139: }
140:      |  RANGE_MAX_SY EQUAL_SY STRING
141: {
142:     variable* temp;
143:     temp = new_variable();
144:     temp->range_max = $3;

```

```

145:         $$ = temp;
146: }
147: |   LENGTH_SY EQUAL_SY STRING
148: {
149:     variable* temp;
150:     temp = new_variable();
151:     temp->length = $3;
152:     $$ = temp;
153: }
154: |   DEFAULT_VALUE_SY EQUAL_SY STRING
155: {
156:     variable* temp;
157:     temp = new_variable();
158:     temp->default_value = $3;
159:     $$ = temp;
160: }
161: |   PRECISION_SY EQUAL_SY STRING
162: {
163:     variable* temp;
164:     temp = new_variable();
165:     temp->precision = $3;
166:     $$ = temp;
167: }
168: |   UNITS_SY EQUAL_SY STRING
169: {
170:     variable* temp;
171:     temp = new_variable();
172:     temp->units = $3;
173:     $$ = temp;
174: }
175: |   ACCURACY_SY EQUAL_SY STRING
176: {
177:     variable* temp;
178:     temp = new_variable();
179:     temp->accuracy = $3;
180:     $$ = temp;
181: }
182: |   SCALE_FACTOR_SY EQUAL_SY STRING
183: {
184:     variable* temp;
185:     temp = new_variable();

```

```

186:         temp->scale_factor = $3;
187:         $$ = temp;
188:     }
189:     |   SCALE_UNITS_SY EQUAL_SY STRING
190: {
191:     variable* temp;
192:     temp = new_variable();
193:     temp->scale_units = $3;
194:     $$ = temp;
195: }
196:     ;
197:
198: VAR_ELEMENTS      :   VAR_QUALIFIERS VAR_SUBELEMENTS
199: {
200:     $$ = merge_variables($1,$2);
201: }
202:     ;
203:
204: VAR_QUALIFIERS    :   QUALIFIERS_SECTION
205: {
206:     variable* temp;
207:     temp = new_variable();
208:     temp->qualifiers = $1;
209:     $$ = temp;
210: }
211:     ;
212:
213:
214: VAR_SUBELEMENTS   :   VAR_SUBELEMENTS VAR_SUBELEMENT
215: {
216:     $$ = merge_variables($1,$2);
217: }
218:     |   /*empty*/
219: {
220:     $$ = NULL;
221: }
222:     ;
223:
224: VAR_SUBELEMENT    :   DRANGE
225: {
226:     variable* temp;

```

```

227:         temp = new_variable();
228:         temp->dranges = $1;
229:         $$ = temp;
230: }
231: | CURVE
232: {
233:         variable* temp;
234:         temp = new_variable();
235:         temp->curves = $1;
236:         $$ = temp;
237: }
238: ;
239:
240: QUALIFIERS      :   QUALIFIERS_WITH_SUBELEMENTS
241: {
242:         $$=$1
243: }
244: | QUALIFIERS_NO_SUBELEMENTS
245: {
246:         $$=$1
247: }
248: ;
249:
250: QUALIFIERS_WITH_SUBELEMENTS : QUALIFIERS_HEAD          CLOSE_SY
QUALIFIERS_SUBELEMENTS CLOSE_QUALIFIER_SY CLOSE_SY
251: {
252:         $$ = $1;
253: }
254: ;
255:
256: QUALIFIERS_NO_SUBELEMENTS : QUALIFIERS_HEAD SLASHCLOSE_SY
257: {
258:         $$ = $1;
259: }
260: ;
261:
262: QUALIFIERS_SUBELEMENTS : /*empty*/
263: ;
264:
265: QUALIFIERS_HEAD      : OPEN_QUALIFIER_SY QUALIFIERS_ATTRIBUTES
266: {

```

```

267:          $$ = $2;
268: }
269:      ;
270:
271: QUALIFIERS_ATTRIBUTES :   QUALIFIERS_ATTRIBUTES QUALIFIERS_ATTRIBUTE
272: {
273:     $$ = merge_qualifiers($1,$2);
274: }
275: |   /*empty*/
276: {
277:     $$ = NULL;
278: }
279:      ;
280:
281: QUALIFIERS_ATTRIBUTE  :   NAME_SY EQUAL_SY STRING
282: {
283:     qualifier_type* temp;
284:     temp = new_qualifier();
285:     temp->name = $3;
286:     $$ = temp;
287: }
288: |   VALUE_SY EQUAL_SY STRING
289: {
290:     qualifier_type* temp;
291:     temp = new_qualifier();
292:     temp->value = $3;
293:     $$ = temp;
294: }
295: |   UNITS_SY EQUAL_SY STRING
296: {
297:     qualifier_type* temp;
298:     temp = new_qualifier();
299:     temp->units = $3;
300:     $$ = temp;
301: }
302:      ;
303:
304: DRANGE      :   DRANGE_HEAD      DRANGE_OPTIONS      CLOSE_DRANGE_SY
CLOSE_SY
305: {
306:     drange* temp;

```



```

307:         temp = new_drangle();
308:         temp->options = $2;
309:         $$ = merge_drangles($1,temp);
310: }
311:     ;
312:
313: DRANGE_HEAD      :   OPEN_DRANGE_SY DRANGE_ATTRIBUTES CLOSE_SY
314: {
315:         $$ = $2;
316: }
317:     ;
318:
319: DRANGE_ATTRIBUTES :   DRANGE_ATTRIBUTES DRANGE_ATTRIBUTE
320: {
321:         $$ = merge_drangles($1,$2);
322: }
323:     |   /*empty*/
324: {
325:         $$ = NULL;
326: }
327:     ;
328:
329: DRANGE_ATTRIBUTE  :   NAME_SY EQUAL_SY STRING
330: {
331:         drangle* temp;
332:         temp = new_drangle();
333:         temp->name = $3;
334:         $$ = temp;
335: }
336:     |   DESCRIPTION_SY EQUAL_SY STRING
337: {
338:         drangle* temp;
339:         temp = new_drangle();
340:         temp->description = $3;
341:         $$ = temp;
342: }
343:     ;
344:
345: DRANGE_OPTIONS    :   DRANGE_OPTIONS DRANGE_OPTION
346: {
347:         $$ = link_options($1,$2);

```

```

348: }
349:     | /*empty*/
350: {
351:     $$ = NULL;
352: }
353:     ;
354:
355: DRANGE_OPTION      :   OPEN_OPTION_SY OPTION_ATTRIBUTES SLASHCLOSE_SY
356: {
357:     $$ = $2;
358: }
359:     |   OPEN_OPTION_SY OPTION_ATTRIBUTES CLOSE_SY CLOSE_OPTION_SY
CLOSE_SY
360: {
361:     $$ = $2;
362: }
363:     ;
364:
365: OPTION_ATTRIBUTES  :   OPTION_ATTRIBUTES OPTION_ATTRIBUTE
366: {
367:     $$ = merge_options($1,$2);
368: }
369:     | /*empty*/
370: {
371:     $$ = NULL;
372: }
373:     ;
374:
375: OPTION_ATTRIBUTE   :   NAME_SY EQUAL_SY STRING
376: {
377:     curveoption* temp;
378:     temp = new_option();
379:     temp->name = $3;
380:     $$ = temp;
381: }
382:     |   VALUE_SY EQUAL_SY STRING
383: {
384:     curveoption* temp;
385:     temp = new_option();
386:     temp->value = $3;
387:     $$ = temp;

```

```

388: }
389:     | DESCRIPTION_SY EQUAL_SY STRING
390: {
391:     curveoption* temp;
392:     temp = new_option();
393:     temp->description = $3;
394:     $$ = temp;
395: }
396:     | ALARM_SY EQUAL_SY STRING
397: {
398:     curveoption* temp;
399:     temp = new_option();
400:     temp->alarm = $3;
401:     $$ = temp;
402: }
403:     ;
404: CURVE          : CURVE_HEAD CURVE_COEFFS CLOSE_CURVE_SY CLOSE_SY
405: {
406:     curve* temp;
407:     temp = new_curve();
408:     temp->coefs = $2;
409:     $$ = merge_curves($1,temp);
410: }
411:     ;
412:
413: CURVE_HEAD      : OPEN_CURVE_SY CURVE_ATTRIBUTES CLOSE_SY
414: {
415:     $$ = $2;
416: }
417:     ;
418:
419: CURVE_ATTRIBUTES : CURVE_ATTRIBUTES CURVE_ATTRIBUTE
420: {
421:     $$ = merge_curves($1,$2);
422: }
423:     | /*empty*/
424: {
425:     $$ = NULL;
426: }
427:     ;
428:

```

```

429: CURVE_ATTRIBUTE      :   NAME_SY EQUAL_SY STRING
430: {
431:     curve* temp;
432:         temp = new_curve();
433:         temp->name = $3;
434:         $$ = temp;
435: }
436:     |   DESCRIPTION_SY EQUAL_SY STRING
437: {
438:     curve* temp;
439:         temp = new_curve();
440:         temp->description = $3;
441:         $$ = temp;
442: }
443:     ;
444:
445: CURVE_COEFFS      :   CURVE_COEFFS CURVE_COEFF
446: {
447:     $$ = link_coefs($1,$2);
448: }
449:     |   /*empty*/
450: {
451:     $$ = NULL;
452: }
453:     ;
454:
455: CURVE_COEFF      :   OPEN_COEFF_SY COEFF_ATTRIBUTES SLASHCLOSE_SY
456: {
457:     $$ = $2;
458: }
459:     |   OPEN_COEFF_SY  COEFF_ATTRIBUTES  CLOSE_SY  CLOSE_COEFF_SY
CLOSE_SY
460: {
461:     $$ = $2;
462: }
463:     |   OPEN_COEFF_SY COEFF_ATTRIBUTES CLOSE_SY
464: {
465:     $$ = $2;
466: }
467:     ;
468:

```

```

469: COEFF_ATTRIBUTES : COEFF_ATTRIBUTES COEFF_ATTRIBUTE
470: {
471:     $$ = merge_coefs($1,$2);
472: }
473:     | /*empty*/
474: {
475:     $$ = NULL;
476: }
477:     ;
478:
479: COEFF_ATTRIBUTE : EXPONENT_SY EQUAL_SY STRING
480: {
481:     coef* temp;
482:         temp = new_coef();
483:         temp->exponent = $3;
484:         $$ = temp;
485: }
486:     | VALUE_SY EQUAL_SY STRING
487: {
488:     coef* temp;
489:         temp = new_coef();
490:         temp->value = $3;
491:         $$ = temp;
492: }
493:     | DESCRIPTION_SY EQUAL_SY STRING
494: {
495:     coef* temp;
496:         temp = new_coef();
497:         temp->description = $3;
498:         $$ = temp;
499: }
500:     ;
501: QUALIFIERS_SECTION : QUALIFIERS_SECTION QUALIFIERS
502: {
503:     $$= link_qualifiers($1,$2);
504: }
505:     | /*empty*/
506: {
507:     $$= NULL;
508: }
509:     ;

```

510:

511: %%

File: sdm/common/VarInfoParser/Makefile

```
1: include ../../Makefile.common
2: include ../../$(MAKEFILE_DEFS)
3:
4: .PHONY: all clean distclean
5:
6: all: lex.VarInfoParser.o VarInfoParser.tab.o Variable.o VarInfoParser.o
7:
8: #rules for the VarInfoParser class
9: VarInfoParser.o: VarInfoParser.cpp VarInfoParser.h Variable.h
10: $(CXX) $(CXXFLAGS) -fPIC -c $<
11:
12: #rules for lexical analyzer
13: lex.VarInfoParser.o: lex.VarInfoParser.c VarInfoParser.tab.c
14: $(CC) $(CFLAGS) -fPIC -c $<
15:
16: lex.VarInfoParser.c: VarInfoParser.l
17: $(LEX) $(LEXFLAGS) -PVarInfoParser $<
18:
19: #rules for parser
20: VarInfoParser.tab.o: VarInfoParser.tab.c
21: $(CC) $(CFLAGS) -fPIC -c $<
22:
23: VarInfoParser.tab.c: VarInfoParser.y
24: $(YACC) $(YACCFLAGS) -p VarInfoParser $<
25:
26: #rules for Variable
27: Variable.o: Variable.c Variable.h
28: $(CC) $(CFLAGS) -fPIC -c $<
29:
30: clean:
31: rm -f *.o *~
32:
33: distclean: clean
```

File: sdm/common/VarInfoParser/VarInfoParser.tab.c

```
1: /* A Bison parser, made by GNU Bison 1.875d. */
2:
3: /* Skeleton parser for Yacc-like parsing with Bison,
4:  Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.
5:
6:  This program is free software; you can redistribute it and/or modify
7:  it under the terms of the GNU General Public License as published by
8:  the Free Software Foundation; either version 2, or (at your option)
9:  any later version.
10:
11:  This program is distributed in the hope that it will be useful,
12:  but WITHOUT ANY WARRANTY; without even the implied warranty of
13:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14:  GNU General Public License for more details.
15:
16:  You should have received a copy of the GNU General Public License
17:  along with this program; if not, write to the Free Software
18:  Foundation, Inc., 59 Temple Place - Suite 330,
19:  Boston, MA 02111-1307, USA. */
20:
21: /* As a special exception, when this file is copied by Bison into a
22:  Bison output file, you may use that output file without restriction.
23:  This special exception was added by the Free Software Foundation
24:  in version 1.24 of Bison. */
25:
26: /* Written by Richard Stallman by simplifying the original so called
27:  ``semantic" parser. */
28:
29: /* All symbols defined below should begin with yy or YY, to avoid
30:  infringing on user name space.  This should be done even for local
31:  variables, as they might otherwise be expanded by user macros.
32:  There are some unavoidable exceptions within include files to
33:  define necessary library symbols; they are noted "INFRINGES ON
34:  USER NAME SPACE" below. */
35:
36: /* Identify Bison output. */
37: #define YYBISON 1
38:
39: /* Skeleton name. */
```



```

40: #define YYSKELETON_NAME "yacc.c"
41:
42: /* Pure parsers. */
43: #define YYPURE 0
44:
45: /* Using locations. */
46: #define YYLSP_NEEDED 0
47:
48: /* If NAME_PREFIX is specified substitute the variables and functions
49:  names. */
50: #define yyparse VarInfoParserparse
51: #define yylex VarInfoParserlex
52: #define yyerror VarInfoParsererror
53: #define yylval VarInfoParserlval
54: #define yychar VarInfoParserchar
55: #define yydebug VarInfoParserdebug
56: #define yynerrs VarInfoParsernerrs
57:
58:
59: /* Tokens. */
60: #ifndef YYTOKENTYPE
61: # define YYTOKENTYPE
62:  /* Put the tokens into the symbol table, so that GDB and other debuggers
63:   know about them. */
64:  enum yytokentype {
65:   EQUAL_SY = 258,
66:   CLOSE_SY = 259,
67:   SLASHCLOSE_SY = 260,
68:   OPEN_XML_SY = 261,
69:   CLOSE_xTEDS_SY = 262,
70:   OPEN_xTEDS_SY = 263,
71:   OPEN_APP_SY = 264,
72:   OPEN_VAR_SY = 265,
73:   CLOSE_VAR_SY = 266,
74:   OPEN_DRANGE_SY = 267,
75:   CLOSE_DRANGE_SY = 268,
76:   OPEN_OPTION_SY = 269,
77:   OPEN_CURVE_SY = 270,
78:   CLOSE_CURVE_SY = 271,
79:   OPEN_COEFF_SY = 272,
80:   OPEN_DATA_MSG_SY = 273,

```

81: CLOSE_DATA_MSG_SY = 274,
82: OPEN_VARIABLE_REF_SY = 275,
83: OPEN_COMMAND_MSG_SY = 276,
84: CLOSE_COMMAND_MSG_SY = 277,
85: NAME_SY = 278,
86: KIND_SY = 279,
87: ID_SY = 280,
88: CLOSE_ORIENTATION_SY = 281,
89: QUALIFIER_SY = 282,
90: DESCRIPTION_SY = 283,
91: MANUFACTURER_ID_SY = 284,
92: VERSION_SY = 285,
93: MODEL_ID_SY = 286,
94: VERSION_LETTER_SY = 287,
95: SERIAL_NUMBER_SY = 288,
96: CALIBRATION_DATE_SY = 289,
97: SENSITIVITY_AT_REF_SY = 290,
98: REF_FREQ_SY = 291,
99: REF_TEMP_SY = 292,
100: MEASUREMENT_RANGE_SY = 293,
101: ELECTRICAL_OUTPUT_SY = 294,
102: QUALITY_FACTOR_SY = 295,
103: TEMP_COEFF_SY = 296,
104: DIRECTION_XYZ_SY = 297,
105: CAL_DUE_DATE_SY = 298,
106: POWER_REQS_SY = 299,
107: VALUE_SY = 300,
108: ALARM_SY = 301,
109: MSG_ARRIVAL_SY = 302,
110: MSG_RATE_SY = 303,
111: STRING = 304,
112: FLOAT = 305,
113: INT = 306,
114: PRECISION_SY = 307,
115: RANGE_MAX_SY = 308,
116: CLOSE_LOCATION_SY = 309,
117: FORMAT_SY = 310,
118: ACCURACY_SY = 311,
119: RANGE_MIN_SY = 312,
120: SCALE_FACTOR_SY = 313,
121: UNITS_SY = 314,

122: DEFAULT_VALUE_SY = 315,
123: OPEN_DEVICE_SY = 316,
124: SCALE_UNITS_SY = 317,
125: LENGTH_SY = 318,
126: EXPONENT_SY = 319,
127: SCHEMA_LOCATION_SY = 320,
128: XMLNS_SY = 321,
129: XMLNS_XSI_SY = 322,
130: CLOSE_OPTION_SY = 323,
131: OPEN_INTERFACE_SY = 324,
132: OPEN_COMMAND_SY = 325,
133: OPEN_NOTIFICATION_SY = 326,
134: OPEN_REQUEST_SY = 327,
135: OPEN_FAULT_MSG_SY = 328,
136: COMPONENT_KEY_SY = 329,
137: SPA_U_HUB_SY = 330,
138: SPA_U_PORT_SY = 331,
139: EXTENDS_SY = 332,
140: CLOSE_COMMAND_SY = 333,
141: CLOSE_NOTIFICATION_SY = 334,
142: CLOSE_REQUEST_SY = 335,
143: CLOSE_FAULT_MSG_SY = 336,
144: OPEN_QUALIFIER_SY = 337,
145: CLOSE_QUALIFIER_SY = 338,
146: CLOSE_APP_SY = 339,
147: CLOSE_DEVICE_SY = 340,
148: CLOSE_INTERFACE_SY = 341,
149: MEMORY_MINIMUM_SY = 342,
150: OPERATING_SYSTEM_SY = 343,
151: PATH_FOR_ASSEMBLY_SY = 344,
152: PATH_ON_SPACECRAFT_SY = 345,
153: X_SY = 346,
154: Y_SY = 347,
155: Z_SY = 348,
156: AXIS_SY = 349,
157: ANGLE_SY = 350,
158: OPEN_LOCATION_SY = 351,
159: OPEN_ORIENTATION_SY = 352,
160: CLOSE_XML_SY = 353,
161: ENCODING_SY = 354,
162: STANDALONE_SY = 355,

163: CLOSE_VARIABLE_REF_SY = 356,
164: CLOSE_COEFF_SY = 357
165: };
166: #endif
167: #define EQUAL_SY 258
168: #define CLOSE_SY 259
169: #define SLASHCLOSE_SY 260
170: #define OPEN_XML_SY 261
171: #define CLOSE_xTEDS_SY 262
172: #define OPEN_xTEDS_SY 263
173: #define OPEN_APP_SY 264
174: #define OPEN_VAR_SY 265
175: #define CLOSE_VAR_SY 266
176: #define OPEN_DRANGE_SY 267
177: #define CLOSE_DRANGE_SY 268
178: #define OPEN_OPTION_SY 269
179: #define OPEN_CURVE_SY 270
180: #define CLOSE_CURVE_SY 271
181: #define OPEN_COEFF_SY 272
182: #define OPEN_DATA_MSG_SY 273
183: #define CLOSE_DATA_MSG_SY 274
184: #define OPEN_VARIABLE_REF_SY 275
185: #define OPEN_COMMAND_MSG_SY 276
186: #define CLOSE_COMMAND_MSG_SY 277
187: #define NAME_SY 278
188: #define KIND_SY 279
189: #define ID_SY 280
190: #define CLOSE_ORIENTATION_SY 281
191: #define QUALIFIER_SY 282
192: #define DESCRIPTION_SY 283
193: #define MANUFACTURER_ID_SY 284
194: #define VERSION_SY 285
195: #define MODEL_ID_SY 286
196: #define VERSION_LETTER_SY 287
197: #define SERIAL_NUMBER_SY 288
198: #define CALIBRATION_DATE_SY 289
199: #define SENSITIVITY_AT_REF_SY 290
200: #define REF_FREQ_SY 291
201: #define REF_TEMP_SY 292
202: #define MEASUREMENT_RANGE_SY 293
203: #define ELECTRICAL_OUTPUT_SY 294

204: #define QUALITY_FACTOR_SY 295
205: #define TEMP_COEFF_SY 296
206: #define DIRECTION_XYZ_SY 297
207: #define CAL_DUE_DATE_SY 298
208: #define POWER_REQS_SY 299
209: #define VALUE_SY 300
210: #define ALARM_SY 301
211: #define MSG_ARRIVAL_SY 302
212: #define MSG_RATE_SY 303
213: #define STRING 304
214: #define FLOAT 305
215: #define INT 306
216: #define PRECISION_SY 307
217: #define RANGE_MAX_SY 308
218: #define CLOSE_LOCATION_SY 309
219: #define FORMAT_SY 310
220: #define ACCURACY_SY 311
221: #define RANGE_MIN_SY 312
222: #define SCALE_FACTOR_SY 313
223: #define UNITS_SY 314
224: #define DEFAULT_VALUE_SY 315
225: #define OPEN_DEVICE_SY 316
226: #define SCALE_UNITS_SY 317
227: #define LENGTH_SY 318
228: #define EXPONENT_SY 319
229: #define SCHEMA_LOCATION_SY 320
230: #define XMLNS_SY 321
231: #define XMLNS_XSI_SY 322
232: #define CLOSE_OPTION_SY 323
233: #define OPEN_INTERFACE_SY 324
234: #define OPEN_COMMAND_SY 325
235: #define OPEN_NOTIFICATION_SY 326
236: #define OPEN_REQUEST_SY 327
237: #define OPEN_FAULT_MSG_SY 328
238: #define COMPONENT_KEY_SY 329
239: #define SPA_U_HUB_SY 330
240: #define SPA_U_PORT_SY 331
241: #define EXTENDS_SY 332
242: #define CLOSE_COMMAND_SY 333
243: #define CLOSE_NOTIFICATION_SY 334
244: #define CLOSE_REQUEST_SY 335

```

245: #define CLOSE_FAULT_MSG_SY 336
246: #define OPEN_QUALIFIER_SY 337
247: #define CLOSE_QUALIFIER_SY 338
248: #define CLOSE_APP_SY 339
249: #define CLOSE_DEVICE_SY 340
250: #define CLOSE_INTERFACE_SY 341
251: #define MEMORY_MINIMUM_SY 342
252: #define OPERATING_SYSTEM_SY 343
253: #define PATH_FOR_ASSEMBLY_SY 344
254: #define PATH_ON_SPACECRAFT_SY 345
255: #define X_SY 346
256: #define Y_SY 347
257: #define Z_SY 348
258: #define AXIS_SY 349
259: #define ANGLE_SY 350
260: #define OPEN_LOCATION_SY 351
261: #define OPEN_ORIENTATION_SY 352
262: #define CLOSE_XML_SY 353
263: #define ENCODING_SY 354
264: #define STANDALONE_SY 355
265: #define CLOSE_VARIABLE_REF_SY 356
266: #define CLOSE_COEFF_SY 357
267:
268:
269:
270:
271: /* Copy the first part of user declarations. */
272: #line 1 "VarInfoParser.y"
273:
274: #include <stdio.h>
275: #include <string.h>
276: #include "../xTEDS/xTEDSParser.h"
277:
278:
279: /* Enabling traces. */
280: #ifndef YYDEBUG
281: # define YYDEBUG 1
282: #endif
283:
284: /* Enabling verbose error messages. */
285: #ifdef YYERROR_VERBOSE

```

```

286: # undef YYERROR_VERBOSE
287: # define YYERROR_VERBOSE 1
288: #else
289: # define YYERROR_VERBOSE 0
290: #endif
291:
292: #if ! defined (YYSTYPE) && ! defined (YYSTYPE_IS_DECLARED)
293: #line 40 "VarInfoParser.y"
294: typedef union YYSTYPE {
295:     int integer;
296:     float real;
297:     char* str;
298:     struct variable_data* var;
299:     struct qualifier_data* qual;
300:     struct coefficient_data* coef;
301:     struct curve_data* curve;
302:     struct option_data* curveoption;
303:     struct drange_data* drange;
304: } YYSTYPE;
305: /* Line 191 of yacc.c. */
306: #line 307 "VarInfoParser.tab.c"
307: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
308: # define YYSTYPE_IS_DECLARED 1
309: # define YYSTYPE_IS_TRIVIAL 1
310: #endif
311:
312:
313:
314: /* Copy the second part of user declarations. */
315:
316:
317: /* Line 214 of yacc.c. */
318: #line 319 "VarInfoParser.tab.c"
319:
320: #if ! defined (yyoverflow) || YYERROR_VERBOSE
321:
322: # ifndef YYFREE
323: #  define YYFREE free
324: # endif
325: # ifndef YYMALLOC
326: #  define YYMALLOC malloc

```

```

327: # endif
328:
329: /* The parser invokes alloca or malloc; define the necessary symbols. */
330:
331: # ifdef YYSTACK_USE_ALLOCA
332: #   if YYSTACK_USE_ALLOCA
333: #     define YYSTACK_ALLOC alloca
334: #   endif
335: # else
336: #   if defined (alloca) || defined (_ALLOCA_H)
337: #     define YYSTACK_ALLOC alloca
338: #   else
339: #     ifdef __GNUC__
340: #       define YYSTACK_ALLOC __builtin_alloca
341: #     endif
342: #   endif
343: # endif
344:
345: # ifdef YYSTACK_ALLOC
346: /* Pacify GCC's 'empty if-body' warning. */
347: #   define YYSTACK_FREE(Ptr) do { /* empty */; } while (0)
348: # else
349: #   if defined (__STDC__) || defined (__cplusplus)
350: #     include <stdlib.h> /* INFRINGES ON USER NAME SPACE */
351: #     define YYSIZE_T size_t
352: #   endif
353: #   define YYSTACK_ALLOC YYMALLOC
354: #   define YYSTACK_FREE YYFREE
355: # endif
356: #endif /* ! defined (yyoverflow) || YYERROR_VERBOSE */
357:
358:
359: #if (! defined (yyoverflow) \
360:    && (! defined (__cplusplus) \
361:      || (defined (YYSTYPE_IS_TRIVIAL) && YYSTYPE_IS_TRIVIAL)))
362:
363: /* A type that is properly aligned for any stack member. */
364: union yyallocl
365: {
366:   short int yyss;
367:   YYSTYPE yyvs;

```



```

368: };
369:
370: /* The size of the maximum gap between one aligned stack and the next. */
371: # define YYSTACK_GAP_MAXIMUM (sizeof (union yyalloc) - 1)
372:
373: /* The size of an array large to enough to hold all stacks, each with
374:  N elements. */
375: # define YYSIZE_BYTES(N) \
376:   ((N) * (sizeof (short int) + sizeof (YYSTYPE)) \
377:    + YYSTACK_GAP_MAXIMUM)
378:
379: /* Copy COUNT objects from FROM to TO. The source and destination do
380:  not overlap. */
381: # ifndef YYCOPY
382: #   if defined (__GNUC__) && 1 < __GNUC__
383: #     define YYCOPY(To, From, Count) \
384:       __builtin_memcpy (To, From, (Count) * sizeof (*(From)))
385: #   else
386: #     define YYCOPY(To, From, Count) \
387:       do \
388:         { \
389:           register YYSIZE_T yyi; \
390:           for (yyi = 0; yyi < (Count); yyi++) \
391:             (To)[yyi] = (From)[yyi]; \
392:         } \
393:       while (0)
394: #   endif
395: # endif
396:
397: /* Relocate STACK from its old location to the new one. The
398:  local variables YYSIZE and YYSTACKSIZE give the old and new number of
399:  elements in the stack, and YYPTR gives the new location of the
400:  stack. Advance YYPTR to a properly aligned location for the next
401:  stack. */
402: # define YYSTACK_RELOCATE(Stack) \
403:   do \
404:     { \
405:       YYSIZE_T yynewbytes; \
406:       YYCOPY (&yyptr->Stack, Stack, yysize); \
407:       Stack = &yyptr->Stack; \
408:       yynewbytes = yystacksize * sizeof (*Stack) + YYSTACK_GAP_MAXIMUM; \

```

```

409:  yyptr += yynewbytes / sizeof (*yyptr);          \
410:  }                                                \
411:  while (0)
412:
413: #endif
414:
415: #if defined (__STDC__) || defined (__cplusplus)
416:  typedef signed char yysigned_char;
417: #else
418:  typedef short int yysigned_char;
419: #endif
420:
421: /* YYFINAL -- State number of the termination state. */
422: #define YYFINAL 7
423: /* YYLAST -- Last index in YYTABLE. */
424: #define YYLAST 115
425:
426: /* YYNTOKENS -- Number of terminals. */
427: #define YYNTOKENS 103
428: /* YYNNTS -- Number of nonterminals. */
429: #define YYNNTS 35
430: /* YYNRULES -- Number of rules. */
431: #define YYNRULES 74
432: /* YYNRULES -- Number of states. */
433: #define YYNSTATES 151
434:
435: /* YYTRANSLATE(YYLEX) -- Bison symbol number corresponding to YYLEX. */
436: #define YYUNDEFTOK 2
437: #define YYMAXUTOK 357
438:
439: #define YYTRANSLATE(YYX)          \
440:  ((unsigned int) (YYX) <= YYMAXUTOK ? yytranslate[YYX] : YYUNDEFTOK)
441:
442: /* YYTRANSLATE[YYLEX] -- Bison symbol number corresponding to YYLEX. */
443: static const unsigned char yytranslate[] =
444: {
445:   0,  2,  2,  2,  2,  2,  2,  2,  2,  2,
446:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
447:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
448:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
449:   2,  2,  2,  2,  2,  2,  2,  2,  2,  2,

```

```

450:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
451:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
452:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
453:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
454:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
455:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
456:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
457:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
458:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
459:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
460:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
461:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
462:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
463:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
464:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
465:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
466:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
467:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
468:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
469:  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
470:  2,  2,  2,  2,  2,  2,  1,  2,  3,  4,
471:  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
472: 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
473: 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
474: 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
475: 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
476: 55, 56, 57, 58, 59, 60, 61, 62, 63, 64,
477: 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
478: 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
479: 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
480: 95, 96, 97, 98, 99, 100, 101, 102
481: };
482:
483: #if YYDEBUG
484: /* YYPRHS[YYN] -- Index of the first RHS symbol of rule number YYN in
485:    YYRHS. */
486: static const unsigned char yyprhs[] =
487: {
488:    0,  0,  3,  5,  7, 13, 16, 19, 22, 23,
489:   27, 31, 35, 39, 43, 47, 51, 55, 59, 63,
490:   67, 71, 75, 79, 83, 86, 88, 91, 92, 94,

```

```

491: 96, 98, 100, 106, 109, 110, 113, 116, 117, 121,
492: 125, 129, 134, 138, 141, 142, 146, 150, 153, 154,
493: 158, 164, 167, 168, 172, 176, 180, 184, 189, 193,
494: 196, 197, 201, 205, 208, 209, 213, 219, 223, 226,
495: 227, 231, 235, 239, 242
496: };
497:
498: /* YYRHS -- A '-1'-separated list of the rules' RHS. */
499: static const short int yyrhs[] =
500: {
501: 104, 0, -1, 105, -1, 106, -1, 107, 4, 110,
502: 11, 4, -1, 107, 5, -1, 10, 108, -1, 108,
503: 109, -1, -1, 23, 3, 49, -1, 24, 3, 49,
504: -1, 55, 3, 49, -1, 27, 3, 49, -1, 25,
505: 3, 49, -1, 28, 3, 49, -1, 57, 3, 49,
506: -1, 53, 3, 49, -1, 63, 3, 49, -1, 60,
507: 3, 49, -1, 52, 3, 49, -1, 59, 3, 49,
508: -1, 56, 3, 49, -1, 58, 3, 49, -1, 62,
509: 3, 49, -1, 111, 112, -1, 137, -1, 112, 113,
510: -1, -1, 121, -1, 129, -1, 115, -1, 116, -1,
511: 118, 4, 117, 83, 4, -1, 118, 5, -1, -1,
512: 82, 119, -1, 119, 120, -1, -1, 23, 3, 49,
513: -1, 45, 3, 49, -1, 59, 3, 49, -1, 122,
514: 125, 13, 4, -1, 12, 123, 4, -1, 123, 124,
515: -1, -1, 23, 3, 49, -1, 28, 3, 49, -1,
516: 125, 126, -1, -1, 14, 127, 5, -1, 14, 127,
517: 4, 68, 4, -1, 127, 128, -1, -1, 23, 3,
518: 49, -1, 45, 3, 49, -1, 28, 3, 49, -1,
519: 46, 3, 49, -1, 130, 133, 16, 4, -1, 15,
520: 131, 4, -1, 131, 132, -1, -1, 23, 3, 49,
521: -1, 28, 3, 49, -1, 133, 134, -1, -1, 17,
522: 135, 5, -1, 17, 135, 4, 102, 4, -1, 17,
523: 135, 4, -1, 135, 136, -1, -1, 64, 3, 49,
524: -1, 45, 3, 49, -1, 28, 3, 49, -1, 137,
525: 114, -1, -1
526: };
527:
528: /* YYRLINE[YYN] -- source line where rule number YYN was defined. */
529: static const unsigned short int yyrline[] =
530: {
531: 0, 55, 55, 59, 65, 71, 74, 80, 85, 90,

```

```

532: 97, 104, 111, 119, 126, 133, 140, 147, 154, 161,
533: 168, 175, 182, 189, 198, 204, 214, 219, 224, 231,
534: 240, 244, 250, 256, 262, 265, 271, 276, 281, 288,
535: 295, 304, 313, 319, 324, 329, 336, 345, 350, 355,
536: 359, 365, 370, 375, 382, 389, 396, 404, 413, 419,
537: 424, 429, 436, 445, 450, 455, 459, 463, 469, 474,
538: 479, 486, 493, 501, 506
539: };
540: #endif
541:
542: #if YYDEBUG || YYERROR_VERBOSE
543: /* YYTNME[SYMBOL-NUM] -- String name of the symbol SYMBOL-NUM.
544:    First, the terminals, then, starting at YYNTOKENS, nonterminals. */
545: static const char *const yytname[] =
546: {
547:  "$end", "error", "$undefined", "EQUAL_SY", "CLOSE_SY", "SLASHCLOSE_SY",
548:  "OPEN_XML_SY", "CLOSE_xTEDS_SY", "OPEN_xTEDS_SY", "OPEN_APP_SY",
549:  "OPEN_VAR_SY", "CLOSE_VAR_SY", "OPEN_DRANGE_SY", "CLOSE_DRANGE_SY",
550:  "OPEN_OPTION_SY", "OPEN_CURVE_SY", "CLOSE_CURVE_SY", "OPEN_COEFF_SY",
551:  "OPEN_DATA_MSG_SY", "CLOSE_DATA_MSG_SY", "OPEN_VARIABLE_REF_SY",
552:  "OPEN_COMMAND_MSG_SY", "CLOSE_COMMAND_MSG_SY", "NAME_SY",
553:  "KIND_SY",
554:  "ID_SY", "CLOSE_ORIENTATION_SY", "QUALIFIER_SY", "DESCRIPTION_SY",
555:  "MANUFACTURER_ID_SY", "VERSION_SY", "MODEL_ID_SY", "VERSION_LETTER_SY",
556:  "SERIAL_NUMBER_SY", "CALIBRATION_DATE_SY", "SENSITIVITY_AT_REF_SY",
557:  "REF_FREQ_SY", "REF_TEMP_SY", "MEASUREMENT_RANGE_SY",
558:  "ELECTRICAL_OUTPUT_SY", "QUALITY_FACTOR_SY", "TEMP_COEFF_SY",
559:  "DIRECTION_XYZ_SY", "CAL_DUE_DATE_SY", "POWER_REQS_SY", "VALUE_SY",
560:  "ALARM_SY", "MSG_ARRIVAL_SY", "MSG_RATE_SY", "STRING", "FLOAT", "INT",
561:  "PRECISION_SY", "RANGE_MAX_SY", "CLOSE_LOCATION_SY", "FORMAT_SY",
562:  "ACCURACY_SY", "RANGE_MIN_SY", "SCALE_FACTOR_SY", "UNITS_SY",
563:  "DEFAULT_VALUE_SY", "OPEN_DEVICE_SY", "SCALE_UNITS_SY", "LENGTH_SY",
564:  "EXPONENT_SY", "SCHEMA_LOCATION_SY", "XMLNS_SY", "XMLNS_XSI_SY",
565:  "CLOSE_OPTION_SY", "OPEN_INTERFACE_SY", "OPEN_COMMAND_SY",
566:  "OPEN_NOTIFICATION_SY", "OPEN_REQUEST_SY", "OPEN_FAULT_MSG_SY",
567:  "COMPONENT_KEY_SY", "SPA_U_HUB_SY", "SPA_U_PORT_SY", "EXTENDS_SY",
568:  "CLOSE_COMMAND_SY", "CLOSE_NOTIFICATION_SY", "CLOSE_REQUEST_SY",
569:  "CLOSE_FAULT_MSG_SY", "OPEN_QUALIFIER_SY", "CLOSE_QUALIFIER_SY",
570:  "CLOSE_APP_SY", "CLOSE_DEVICE_SY", "CLOSE_INTERFACE_SY",
571:  "MEMORY_MINIMUM_SY", "OPERATING_SYSTEM_SY", "PATH_FOR_ASSEMBLY_SY",
572:  "PATH_ON_SPACECRAFT_SY", "X_SY", "Y_SY", "Z_SY", "AXIS_SY", "ANGLE_SY",

```

```

572:      "OPEN_LOCATION_SY",    "OPEN_ORIENTATION_SY",    "CLOSE_XML_SY",
"ENCODING_SY",
573: "STANDALONE_SY", "CLOSE_VARIABLE_REF_SY", "CLOSE_COEFF_SY", "$accept",
574:      "VARIABLE",    "VAR_WITH_SUBELEMENTS",    "VAR_NO_SUBELEMENTS",
"VAR_HEAD",
575: "VAR_ATTRIBUTES", "VAR_ATTRIBUTE", "VAR_ELEMENTS", "VAR_QUALIFIERS",
576: "VAR_SUBELEMENTS", "VAR_SUBELEMENT", "QUALIFIERS",
577: "QUALIFIERS_WITH_SUBELEMENTS", "QUALIFIERS_NO_SUBELEMENTS",
578: "QUALIFIERS_SUBELEMENTS", "QUALIFIERS_HEAD", "QUALIFIERS_ATTRIBUTES",
579: "QUALIFIERS_ATTRIBUTE", "DRANGE", "DRANGE_HEAD", "DRANGE_ATTRIBUTES",
580: "DRANGE_ATTRIBUTE", "DRANGE_OPTIONS", "DRANGE_OPTION",
581: "OPTION_ATTRIBUTES", "OPTION_ATTRIBUTE", "CURVE", "CURVE_HEAD",
582: "CURVE_ATTRIBUTES", "CURVE_ATTRIBUTE", "CURVE_COEFFS", "CURVE_COEFF",
583: "COEFF_ATTRIBUTES", "COEFF_ATTRIBUTE", "QUALIFIERS_SECTION", 0
584: };
585: #endif
586:
587: # ifdef YYPRINT
588: /* YYTOKNUM[YYLEX-NUM] -- Internal token number corresponding to
589:  token YYLEX-NUM. */
590: static const unsigned short int yytoknum[] =
591: {
592:     0, 256, 257, 258, 259, 260, 261, 262, 263, 264,
593:     265, 266, 267, 268, 269, 270, 271, 272, 273, 274,
594:     275, 276, 277, 278, 279, 280, 281, 282, 283, 284,
595:     285, 286, 287, 288, 289, 290, 291, 292, 293, 294,
596:     295, 296, 297, 298, 299, 300, 301, 302, 303, 304,
597:     305, 306, 307, 308, 309, 310, 311, 312, 313, 314,
598:     315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
599:     325, 326, 327, 328, 329, 330, 331, 332, 333, 334,
600:     335, 336, 337, 338, 339, 340, 341, 342, 343, 344,
601:     345, 346, 347, 348, 349, 350, 351, 352, 353, 354,
602:     355, 356, 357
603: };
604: # endif
605:
606: /* YYR1[YYN] -- Symbol number of symbol that rule YYN derives. */
607: static const unsigned char yyr1[] =
608: {
609:     0, 103, 104, 104, 105, 106, 107, 108, 108, 109,
610:     109, 109, 109, 109, 109, 109, 109, 109, 109, 109,

```

```

611: 109, 109, 109, 109, 110, 111, 112, 112, 113, 113,
612: 114, 114, 115, 116, 117, 118, 119, 119, 120, 120,
613: 120, 121, 122, 123, 123, 124, 124, 125, 125, 126,
614: 126, 127, 127, 128, 128, 128, 128, 129, 130, 131,
615: 131, 132, 132, 133, 133, 134, 134, 134, 135, 135,
616: 136, 136, 136, 137, 137
617: };
618:
619: /* YYR2[YYN] -- Number of symbols composing right hand side of rule YYN. */
620: static const unsigned char yyr2[] =
621: {
622: 0, 2, 1, 1, 5, 2, 2, 2, 0, 3,
623: 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
624: 3, 3, 3, 3, 2, 1, 2, 0, 1, 1,
625: 1, 1, 5, 2, 0, 2, 2, 0, 3, 3,
626: 3, 4, 3, 2, 0, 3, 3, 2, 0, 3,
627: 5, 2, 0, 3, 3, 3, 3, 4, 3, 2,
628: 0, 3, 3, 2, 0, 3, 5, 3, 2, 0,
629: 3, 3, 3, 2, 0
630: };
631:
632: /* YYDEFAC[STATE-NAME] -- Default rule to reduce with in state
633: STATE-NAME when YYTABLE doesn't specify something else to do. Zero
634: means the default is an error. */
635: static const unsigned char yydefact[] =
636: {
637: 0, 8, 0, 2, 3, 0, 6, 1, 74, 5,
638: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
639: 0, 0, 0, 0, 0, 7, 0, 27, 25, 0,
640: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
641: 0, 0, 0, 0, 0, 24, 37, 73, 30, 31,
642: 0, 9, 10, 13, 12, 14, 19, 16, 11, 21,
643: 15, 22, 20, 18, 23, 17, 4, 44, 60, 26,
644: 28, 48, 29, 64, 35, 34, 33, 0, 0, 0,
645: 0, 0, 0, 0, 36, 0, 42, 0, 0, 43,
646: 58, 0, 0, 59, 0, 52, 47, 0, 69, 63,
647: 0, 0, 0, 0, 0, 0, 0, 0, 41, 0,
648: 57, 0, 38, 39, 40, 32, 45, 46, 61, 62,
649: 0, 49, 0, 0, 0, 0, 51, 67, 65, 0,
650: 0, 0, 68, 0, 0, 0, 0, 0, 0, 0,
651: 0, 0, 50, 53, 55, 54, 56, 66, 72, 71,

```

```

652:    70
653: };
654:
655: /* YYDEFGOTO[NTERM-NUM]. */
656: static const short int yydefgoto[] =
657: {
658:    -1,  2,  3,  4,  5,  6, 25, 26, 27, 45,
659:    69, 47, 48, 49, 85, 50, 74, 84, 70, 71,
660:    77, 89, 79, 96, 109, 126, 72, 73, 78, 93,
661:    80, 99, 111, 132, 28
662: };
663:
664: /* YYPACT[STATE-NUM] -- Index in YYTABLE of the portion describing
665:    STATE-NUM. */
666: #define YYPACT_NINF -66
667: static const yysigned_char yypact[] =
668: {
669:    -1, -66,  5, -66, -66,  7, -5, -66, -66, -66,
670:    22, 30, 32, 33, 34, 35, 36, 37, 42, 43,
671:    46, 53, 56, 58, 59, -66, 52, -66, -65, 15,
672:    16, 17, 18, 19, 20, 21, 23, 24, 25, 26,
673:    27, 28, 29, 31, 67, -8, -66, -66, -66, -66,
674:    9, -66, -66, -66, -66, -66, -66, -66, -66, -66,
675:   -66, -66, -66, -66, -66, -66, -66, -66, -66, -66,
676:   -66, -66, -66, -66, -17, -66, -66,  4,  6,  2,
677:    14, 76, 78, 79, -66,  0, -66, 81, 82, -66,
678:   -66, 83, 84, -66, 85, -66, -66, 86, -66, -66,
679:    39, 44, 45, 87, 47, 48, 49, 50, -66, -2,
680:   -66, -4, -66, -66, -66, -66, -66, -66, -66, -66,
681:    38, -66, 89, 92, 97, 98, -66,  1, -66, 99,
682:   101, 102, -66, 103, 60, 61, 62, 63, 104, 64,
683:    65, 66, -66, -66, -66, -66, -66, -66, -66, -66,
684:   -66
685: };
686:
687: /* YYPGOTO[NTERM-NUM]. */
688: static const yysigned_char yypgoto[] =
689: {
690:   -66, -66, -66, -66, -66, -66, -66, -66, -66, -66,
691:   -66, -66, -66, -66, -66, -66, -66, -66, -66, -66,
692:   -66, -66, -66, -66, -66, -66, -66, -66, -66, -66,

```



```

693:  -66, -66, -66, -66, -66
694: };
695:
696: /* YYTABLE[YYPACT[STATE-NUM]]. What to do in state STATE-NUM. If
697:  positive, shift that token. If negative, reduce the rule which
698:  number is the opposite. If zero, do what YYDEFACCT says.
699:  If YYTABLE_NINF, syntax error. */
700: #define YYTABLE_NINF -1
701: static const unsigned char yytable[] =
702: {
703:  127, 128, 120, 121, 67, 7, 81, 68, 86, 1,
704:  90, 8, 9, 75, 76, 94, 95, 46, 10, 11,
705:  12, 122, 13, 14, 129, 29, 123, 87, 82, 91,
706:  97, 98, 88, 30, 92, 31, 32, 33, 34, 35,
707:  36, 130, 83, 124, 125, 37, 38, 15, 16, 39,
708:  17, 18, 19, 20, 21, 22, 40, 23, 24, 41,
709:  131, 42, 43, 44, 51, 52, 53, 54, 55, 56,
710:  57, 66, 58, 59, 60, 61, 62, 63, 64, 100,
711:  65, 101, 102, 103, 104, 105, 106, 107, 112, 108,
712:  110, 115, 134, 113, 114, 135, 116, 117, 118, 119,
713:  136, 137, 139, 138, 140, 141, 133, 142, 147, 143,
714:  144, 145, 146, 148, 149, 150
715: };
716:
717: static const unsigned char yycheck[] =
718: {
719:  4, 5, 4, 5, 12, 0, 23, 15, 4, 10,
720:  4, 4, 5, 4, 5, 13, 14, 82, 23, 24,
721:  25, 23, 27, 28, 28, 3, 28, 23, 45, 23,
722:  16, 17, 28, 3, 28, 3, 3, 3, 3, 3,
723:  3, 45, 59, 45, 46, 3, 3, 52, 53, 3,
724:  55, 56, 57, 58, 59, 60, 3, 62, 63, 3,
725:  64, 3, 3, 11, 49, 49, 49, 49, 49, 49,
726:  49, 4, 49, 49, 49, 49, 49, 49, 49, 3,
727:  49, 3, 3, 83, 3, 3, 3, 3, 49, 4,
728:  4, 4, 3, 49, 49, 3, 49, 49, 49, 49,
729:  3, 3, 3, 102, 3, 3, 68, 4, 4, 49,
730:  49, 49, 49, 49, 49, 49
731: };
732:
733: /* YYSTOS[STATE-NUM] -- The (internal number of the) accessing

```

```

734: symbol of state STATE-NUM. */
735: static const unsigned char yystos[] =
736: {
737:     0, 10, 104, 105, 106, 107, 108, 0, 4, 5,
738:     23, 24, 25, 27, 28, 52, 53, 55, 56, 57,
739:     58, 59, 60, 62, 63, 109, 110, 111, 137, 3,
740:     3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
741:     3, 3, 3, 3, 11, 112, 82, 114, 115, 116,
742:     118, 49, 49, 49, 49, 49, 49, 49, 49, 49,
743:     49, 49, 49, 49, 49, 49, 4, 12, 15, 113,
744:     121, 122, 129, 130, 119, 4, 5, 123, 131, 125,
745:     133, 23, 45, 59, 120, 117, 4, 23, 28, 124,
746:     4, 23, 28, 132, 13, 14, 126, 16, 17, 134,
747:     3, 3, 3, 83, 3, 3, 3, 3, 4, 127,
748:     4, 135, 49, 49, 49, 4, 49, 49, 49, 49,
749:     4, 5, 23, 28, 45, 46, 128, 4, 5, 28,
750:     45, 64, 136, 68, 3, 3, 3, 3, 102, 3,
751:     3, 3, 4, 49, 49, 49, 49, 4, 49, 49,
752:     49
753: };
754:
755: #if ! defined (YYSIZE_T) && defined (__SIZE_TYPE__)
756: # define YYSIZE_T __SIZE_TYPE__
757: #endif
758: #if ! defined (YYSIZE_T) && defined (size_t)
759: # define YYSIZE_T size_t
760: #endif
761: #if ! defined (YYSIZE_T)
762: # if defined (__STDC__) || defined (__cplusplus)
763: # include <stddef.h> /* INFRINGES ON USER NAME SPACE */
764: # define YYSIZE_T size_t
765: # endif
766: #endif
767: #if ! defined (YYSIZE_T)
768: # define YYSIZE_T unsigned int
769: #endif
770:
771: #define yyerrok      (yyerrstatus = 0)
772: #define yyclearin    (yychar = YYEMPTY)
773: #define YYEMPTY      (-2)
774: #define YYEOF        0

```

```

775:
776: #define YYACCEPT goto yyacceptlab
777: #define YYABORT   goto yyabortlab
778: #define YYERROR   goto yyerrorlab
779:
780:
781: /* Like YYERROR except do call yyerror. This remains here temporarily
782:  to ease the transition to the new meaning of YYERROR, for GCC.
783:  Once GCC version 2 has supplanted version 1, this can go. */
784:
785: #define YYFAIL     goto yyerrlab
786:
787: #define YYRECOVERING() (!yyerrstatus)
788:
789: #define YYBACKUP(Token, Value)          \
790: do                                     \
791:   if (yychar == YYEMPTY && yylen == 1) \
792:   {                                     \
793:     yychar = (Token);                  \
794:     yylval = (Value);                  \
795:     yytoken = YYTRANSLATE (yychar);    \
796:     YYPOPSTACK;                        \
797:     goto yybackup;                     \
798:   }                                     \
799:   else                                 \
800:   {                                     \
801:     yyerror ("syntax error: cannot back up"); \
802:     YYERROR;                             \
803:   }                                     \
804: while (0)
805:
806: #define YYTERROR 1
807: #define YYERRCODE 256
808:
809: /* YYLLOC_DEFAULT -- Compute the default location (before the actions
810:  are run). */
811:
812: #ifndef YYLLOC_DEFAULT
813: # define YYLLOC_DEFAULT(Current, Rhs, N)      \
814:   ((Current).first_line = (Rhs)[1].first_line, \
815:    (Current).first_column = (Rhs)[1].first_column, \

```

```

816: (Current).last_line = (Rhs)[N].last_line, \
817: (Current).last_column = (Rhs)[N].last_column)
818: #endif
819:
820: /* YYLEX -- calling `yylex' with the right arguments. */
821:
822: #ifdef YYLEX_PARAM
823: # define YYLEX yylex (YYLEX_PARAM)
824: #else
825: # define YYLEX yylex ()
826: #endif
827:
828: /* Enable debugging if requested. */
829: #if YYDEBUG
830:
831: # ifndef YYFPRINTF
832: #  include <stdio.h> /* INFRINGES ON USER NAME SPACE */
833: #  define YYFPRINTF fprintf
834: # endif
835:
836: # define YYDPRINTF(Args) \
837: do { \
838:   if (yydebug) \
839:     YYFPRINTF Args; \
840: } while (0)
841:
842: # define YYDSYMPRINT(Args) \
843: do { \
844:   if (yydebug) \
845:     yysymprint Args; \
846: } while (0)
847:
848: # define YYDSYMPRINTF(Title, Token, Value, Location) \
849: do { \
850:   if (yydebug) \
851:   { \
852:     YYFPRINTF (stderr, "%s ", Title); \
853:     yysymprint (stderr, \
854:                 Token, Value); \
855:     YYFPRINTF (stderr, "\n"); \
856:   } \

```

```

857: } while (0)
858:
859: /*-----.
860: | yy_stack_print -- Print the state stack from its BOTTOM up to its |
861: | TOP (included).                                     |
862: `-----*/
863:
864: #if defined (__STDC__) || defined (__cplusplus)
865: static void
866: yy_stack_print (short int *bottom, short int *top)
867: #else
868: static void
869: yy_stack_print (bottom, top)
870:   short int *bottom;
871:   short int *top;
872: #endif
873: {
874:   YYFPRINTF (stderr, "Stack now");
875:   for (/* Nothing. */; bottom <= top; ++bottom)
876:     YYFPRINTF (stderr, " %d", *bottom);
877:   YYFPRINTF (stderr, " \n");
878: }
879:
880: # define YY_STACK_PRINT(Bottom, Top)           \
881: do {                                           \
882:   if (yydebug)                               \
883:     yy_stack_print ((Bottom), (Top));         \
884: } while (0)
885:
886:
887: /*-----.
888: | Report that the YYRULE is going to be reduced. |
889: `-----*/
890:
891: #if defined (__STDC__) || defined (__cplusplus)
892: static void
893: yy_reduce_print (int yyrule)
894: #else
895: static void
896: yy_reduce_print (yyrule)
897:   int yyrule;

```

```

898: #endif
899: {
900:   int yyi;
901:   unsigned int yyno = yyrule[yyrline];
902:   YYFPRINTF (stderr, "Reducing stack by rule %d (line %u), ",
903:             yyrule - 1, yyno);
904:   /* Print the symbols being reduced, and their result. */
905:   for (yyi = yyrhs[yyrule]; 0 <= yyrhs[yyi]; yyi++)
906:     YYFPRINTF (stderr, "%s ", yytname [yyrhs[yyi]]);
907:   YYFPRINTF (stderr, "-> %s \n", yytname [yyr1[yyrule]]);
908: }
909:
910: # define YY_REDUCE_PRINT(Rule)      \
911: do {                                \
912:   if (yydebug)                      \
913:     yy_reduce_print (Rule);        \
914: } while (0)
915:
916: /* Nonzero means print parse trace.  It is left uninitialized so that
917:    multiple parsers can coexist. */
918: int yydebug;
919: #else /* !YYDEBUG */
920: # define YYDPRINTF(Args)
921: # define YYDSYMPRINT(Args)
922: # define YYDSYMPRINTF(Title, Token, Value, Location)
923: # define YY_STACK_PRINT(Bottom, Top)
924: # define YY_REDUCE_PRINT(Rule)
925: #endif /* !YYDEBUG */
926:
927:
928: /* YYINITDEPTH -- initial size of the parser's stacks. */
929: #ifndef YYINITDEPTH
930: # define YYINITDEPTH 200
931: #endif
932:
933: /* YYMAXDEPTH -- maximum size the stacks can grow to (effective only
934:    if the built-in stack extension method is used).
935:
936:    Do not make this value too large; the results are undefined if
937:    SIZE_MAX < YYSTACK_BYTES (YYMAXDEPTH)
938:    evaluated with infinite-precision integer arithmetic. */

```

```

939:
940: #if defined (YYMAXDEPTH) && YYMAXDEPTH == 0
941: # undef YYMAXDEPTH
942: #endif
943:
944: #ifndef YYMAXDEPTH
945: # define YYMAXDEPTH 10000
946: #endif
947:
948:
949:
950: #if YYERROR_VERBOSE
951:
952: # ifndef yystrlen
953: #  if defined (__GLIBC__) && defined (_STRING_H)
954: #   define yystrlen strlen
955: #  else
956: /* Return the length of YYSR. */
957: static YYSIZE_T
958: #  if defined (__STDC__) || defined (__cplusplus)
959: yystrlen (const char *yystr)
960: #  else
961: yystrlen (yystr)
962:   const char *yystr;
963: #  endif
964: {
965:   register const char *yys = yystr;
966:
967:   while (*yys++ != '\0')
968:     continue;
969:
970:   return YYS - yystr - 1;
971: }
972: # endif
973: # endif
974:
975: # ifndef yystpcpy
976: #  if defined (__GLIBC__) && defined (_STRING_H) && defined (_GNU_SOURCE)
977: #   define yystpcpy stpcpy
978: #  else
979: /* Copy YYSRC to YYDEST, returning the address of the terminating '\0' in

```

```

980: YYDEST. */
981: static char *
982: # if defined (__STDC__) || defined (__cplusplus)
983: yystpcpy (char *yydest, const char *yysrc)
984: # else
985: yystpcpy (yydest, yysrc)
986:   char *yydest;
987:   const char *yysrc;
988: # endif
989: {
990:   register char *yyd = yydest;
991:   register const char *yys = yysrc;
992:
993:   while ((*yyd++ = *yys++) != '\0')
994:     continue;
995:
996:   return yyd - 1;
997: }
998: # endif
999: # endif
1000:
1001: #endif /* !YYERROR_VERBOSE */
1002:
1003:
1004:
1005: #if YYDEBUG
1006: /*-----
1007: | Print this symbol on YYOUTPUT. |
1008: `-----*/
1009:
1010: #if defined (__STDC__) || defined (__cplusplus)
1011: static void
1012: yysymprint (FILE *yyoutput, int yytype, YYSTYPE *yyvaluep)
1013: #else
1014: static void
1015: yysymprint (yyoutput, yytype, yyvaluep)
1016:   FILE *yyoutput;
1017:   int yytype;
1018:   YYSTYPE *yyvaluep;
1019: #endif
1020: {

```



```

1021: /* Pacify ``unused variable" warnings. */
1022: (void) yyvaluep;
1023:
1024: if (yytype < YYNTOKENS)
1025: {
1026:     YYFPRINTF (yyoutput, "token %s (", yyname[yytype]);
1027: # ifdef YYPRINT
1028:     YYPRINT (yyoutput, yytoknum[yytype], *yyvaluep);
1029: # endif
1030: }
1031: else
1032:     YYFPRINTF (yyoutput, "nterm %s (", yyname[yytype]);
1033:
1034: switch (yytype)
1035: {
1036:     default:
1037:         break;
1038: }
1039: YYFPRINTF (yyoutput, ")");
1040: }
1041:
1042: #endif /* ! YYDEBUG */
1043: /*-----
1044: | Release the memory associated to this symbol. |
1045: `-----*/
1046:
1047: #if defined (__STDC__) || defined (__cplusplus)
1048: static void
1049: yydestruct (int yytype, YYSTYPE *yyvaluep)
1050: #else
1051: static void
1052: yydestruct (yytype, yyvaluep)
1053:     int yytype;
1054:     YYSTYPE *yyvaluep;
1055: #endif
1056: {
1057:     /* Pacify ``unused variable" warnings. */
1058:     (void) yyvaluep;
1059:
1060:     switch (yytype)
1061:     {

```

```

1062:
1063:     default:
1064:         break;
1065:     }
1066: }
1067:
1068:
1069: /* Prevent warnings from -Wmissing-prototypes. */
1070:
1071: #ifdef YYPARSE_PARAM
1072: # if defined (__STDC__) || defined (__cplusplus)
1073: int yyparse (void *YYPARSE_PARAM);
1074: # else
1075: int yyparse ();
1076: # endif
1077: #else /* ! YYPARSE_PARAM */
1078: #if defined (__STDC__) || defined (__cplusplus)
1079: int yyparse (void);
1080: #else
1081: int yyparse ();
1082: #endif
1083: #endif /* ! YYPARSE_PARAM */
1084:
1085:
1086:
1087: /* The lookahead symbol. */
1088: int yychar;
1089:
1090: /* The semantic value of the lookahead symbol. */
1091: YYSTYPE yylval;
1092:
1093: /* Number of syntax errors so far. */
1094: int yynerrs;
1095:
1096:
1097:
1098: /*-----
1099: | yyparse. |
1100: `-----*/
1101:
1102: #ifdef YYPARSE_PARAM

```

```

1103: # if defined (__STDC__) || defined (__cplusplus)
1104: int yyparse (void *YYPARSE_PARAM)
1105: # else
1106: int yyparse (YYPARSE_PARAM)
1107: void *YYPARSE_PARAM;
1108: # endif
1109: #else /* ! YYPARSE_PARAM */
1110: #if defined (__STDC__) || defined (__cplusplus)
1111: int
1112: yyparse (void)
1113: #else
1114: int
1115: yyparse ()
1116:
1117: #endif
1118: #endif
1119: {
1120:
1121: register int yystate;
1122: register int yyn;
1123: int yyresult;
1124: /* Number of tokens to shift before error messages enabled. */
1125: int yyerrstatus;
1126: /* Lookahead token as an internal (translated) token number. */
1127: int yytoken = 0;
1128:
1129: /* Three stacks and their tools:
1130:  `yyss': related to states,
1131:  `yyvs': related to semantic values,
1132:  `yyls': related to locations.
1133:
1134: Refer to the stacks thru separate pointers, to allow yyoverflow
1135: to reallocate them elsewhere. */
1136:
1137: /* The state stack. */
1138: short int yyssa[YYINITDEPTH];
1139: short int *yyss = yyssa;
1140: register short int *yyssp;
1141:
1142: /* The semantic value stack. */
1143: YYSTYPE yyvsa[YYINITDEPTH];

```

```

1144: YYSTYPE *yyvs = yyvsa;
1145: register YYSTYPE *yyvsp;
1146:
1147:
1148:
1149: #define YYPOPSTACK (yyvsp--, yyssp--)
1150:
1151: YYSIZE_T yystacksize = YYINITDEPTH;
1152:
1153: /* The variables used to return semantic value and location from the
1154:    action routines. */
1155: YYSTYPE yyval;
1156:
1157:
1158: /* When reducing, the number of symbols on the RHS of the reduced
1159:    rule. */
1160: int yylen;
1161:
1162: YYDPRINTF ((stderr, "Starting parse \n"));
1163:
1164: yynstate = 0;
1165: yyerrstatus = 0;
1166: yynerrs = 0;
1167: yychar = YYEMPTY;      /* Cause a token to be read. */
1168:
1169: /* Initialize stack pointers.
1170:    Waste one element of value and location stack
1171:    so that they stay on the same level as the state stack.
1172:    The wasted elements are never initialized. */
1173:
1174: yyssp = yyss;
1175: yyvsp = yyvs;
1176:
1177:
1178: goto yysetstate;
1179:
1180: /*-----
1181: | yynstate -- Push a new state, which is found in yynstate. |
1182: `-----*/
1183: yynstate:
1184: /* In all cases, when you get here, the value and location stacks

```

```

1185:   have just been pushed. so pushing a state here evens the stacks.
1186:   */
1187:   yyssp++;
1188:
1189: yysetstate:
1190:   *yyssp = yystate;
1191:
1192:   if (yyss + yystacksize - 1 <= yyssp)
1193:   {
1194:       /* Get the current used size of the three stacks, in elements. */
1195:       YYSIZE_T yysize = yyssp - yyss + 1;
1196:
1197: #ifdef yyoverflow
1198:     {
1199:       /* Give user a chance to reallocate the stack. Use copies of
1200:        these so that the &'s don't force the real ones into
1201:        memory. */
1202:       YYSTYPE *yyvs1 = yyvs;
1203:       short int *yyss1 = yyss;
1204:
1205:
1206:       /* Each stack pointer address is followed by the size of the
1207:        data in use in that stack, in bytes. This used to be a
1208:        conditional around just the two extra args, but that might
1209:        be undefined if yyoverflow is a macro. */
1210:       yyoverflow ("parser stack overflow",
1211:                  &yyss1, yysize * sizeof (*yyssp),
1212:                  &yyvs1, yysize * sizeof (*yvsp),
1213:
1214:                  &yystacksize);
1215:
1216:       yyss = yyss1;
1217:       yyvs = yyvs1;
1218:     }
1219: #else /* no yyoverflow */
1220: #ifndef YYSTACK_RELOCATE
1221:     goto yyoverflowlab;
1222: # else
1223:     /* Extend the stack our own way. */
1224:     if (YYMAXDEPTH <= yystacksize)
1225:       goto yyoverflowlab;

```

```

1226:    yystacksize *= 2;
1227:    if (YYMAXDEPTH < yystacksize)
1228:        yystacksize = YYMAXDEPTH;
1229:
1230:    {
1231:        short int *yyss1 = yyss;
1232:        union yyalloc *yyptr =
1233:            (union yyalloc *) YYSTACK_ALLOC (YYSTACK_BYTES (ystacksize));
1234:        if (! yyptr)
1235:            goto yyoverflowlab;
1236:        YYSTACK_RELOCATE (yyss);
1237:        YYSTACK_RELOCATE (yyvs);
1238:
1239: # undef YYSTACK_RELOCATE
1240:        if (yyss1 != yyssa)
1241:            YYSTACK_FREE (yyss1);
1242:    }
1243: # endif
1244: #endif /* no yyoverflow */
1245:
1246:    yyssp = yyss + yysize - 1;
1247:    yyvsp = yyvs + yysize - 1;
1248:
1249:
1250:    YYDPRINTF ((stderr, "Stack size increased to %lu \n",
1251:        (unsigned long int) yystacksize));
1252:
1253:    if (yyss + yystacksize - 1 <= yyssp)
1254:        YYABORT;
1255:    }
1256:
1257:    YYDPRINTF ((stderr, "Entering state %d \n", yystate));
1258:
1259:    goto yybackup;
1260:
1261: /*-----,
1262: |yybackup. |
1263: `-----*/
1264: yybackup:
1265:
1266: /* Do appropriate processing given the current state. */

```

```

1267: /* Read a lookahead token if we need one and don't already have one. */
1268: /* yyresume: */
1269:
1270: /* First try to decide what to do without reference to lookahead token. */
1271:
1272: yyn = yypact[yystate];
1273: if (yyn == YYPACT_NINF)
1274:     goto yydefault;
1275:
1276: /* Not known => get a lookahead token if don't already have one. */
1277:
1278: /* YYCHAR is either YYEMPTY or YYEOF or a valid lookahead symbol. */
1279: if (yychar == YYEMPTY)
1280:     {
1281:         YYDPRINTF ((stderr, "Reading a token: "));
1282:         yychar = YYLEX;
1283:     }
1284:
1285: if (yychar <= YYEOF)
1286:     {
1287:         yychar = yytoken = YYEOF;
1288:         YYDPRINTF ((stderr, "Now at end of input. \n"));
1289:     }
1290: else
1291:     {
1292:         yytoken = YYTRANSLATE (yychar);
1293:         YYDSYMPRINTF ("Next token is", yytoken, &yylval, &yyllloc);
1294:     }
1295:
1296: /* If the proper action on seeing token YYTOKEN is to reduce or to
1297:    detect an error, take that action. */
1298: yyn += yytoken;
1299: if (yyn < 0 || YYLAST < yyn || yycheck[yyn] != yytoken)
1300:     goto yydefault;
1301: yyn = yytable[yyn];
1302: if (yyn <= 0)
1303:     {
1304:         if (yyn == 0 || yyn == YYTABLE_NINF)
1305:             goto yyerrlab;
1306:         yyn = -yyn;
1307:         goto yyreduce;

```

```

1308:  }
1309:
1310:  if (yyn == YYFINAL)
1311:    YYACCEPT;
1312:
1313:  /* Shift the lookahead token. */
1314:  YYDPRINTF ((stderr, "Shifting token %s, ", yytname[yytoken]));
1315:
1316:  /* Discard the token being shifted unless it is eof. */
1317:  if (yychar != YYEOF)
1318:    yychar = YYEMPTY;
1319:
1320:  *++yyvsp = yylval;
1321:
1322:
1323:  /* Count tokens shifted since error; after three, turn off error
1324:     status. */
1325:  if (yyerrstatus)
1326:    yyerrstatus--;
1327:
1328:  yystate = yyn;
1329:  goto yynewstate;
1330:
1331:
1332: /*-----
1333: | yydefault -- do the default action for the current state. |
1334: `-----*/
1335: yydefault:
1336:   yyn = yydefact[yystate];
1337:   if (yyn == 0)
1338:     goto yyerrlab;
1339:   goto yyreduce;
1340:
1341:
1342: /*-----
1343: | yyreduce -- Do a reduction. |
1344: `-----*/
1345: yyreduce:
1346:   /* yyn is the number of a rule to reduce with. */
1347:   yrlen = yyr2[yyn];
1348:

```



```

1349: /* If YYLEN is nonzero, implement the default value of the action:
1350:   `$$ = $1'.
1351:
1352:   Otherwise, the following line sets YYVAL to garbage.
1353:   This behavior is undocumented and Bison
1354:   users should not rely upon it. Assigning to YYVAL
1355:   unconditionally makes the parser a bit smaller, and it avoids a
1356:   GCC warning that YYVAL may be used uninitialized. */
1357: yyval = yyvsp[1-yylen];
1358:
1359:
1360: YY_REDUCE_PRINT (yyn);
1361: switch (yyn)
1362: {
1363:   case 2:
1364: #line 56 "VarInfoParser.y"
1365:   {
1366:       yyval.var=yyvsp[0].var
1367:   ;}
1368:   break;
1369:
1370:   case 3:
1371: #line 60 "VarInfoParser.y"
1372:   {
1373:       yyval.var=yyvsp[0].var
1374:   ;}
1375:   break;
1376:
1377:   case 4:
1378: #line 66 "VarInfoParser.y"
1379:   {
1380:       yyval.var = merge_variables(yyvsp[-4].var,yyvsp[-2].var);
1381:   ;}
1382:   break;
1383:
1384:   case 6:
1385: #line 75 "VarInfoParser.y"
1386:   {
1387:       yyval.var= yyvsp[0].var
1388:   ;}
1389:   break;

```

```

1390:
1391: case 7:
1392: #line 81 "VarInfoParser.y"
1393: {
1394:         yyval.var= merge_variables(yyvsp[-1].var,yyvsp[0].var);
1395: ;}
1396: break;
1397:
1398: case 8:
1399: #line 85 "VarInfoParser.y"
1400: {
1401:         yyval.var=NULL;
1402: ;}
1403: break;
1404:
1405: case 9:
1406: #line 91 "VarInfoParser.y"
1407: {
1408:         variable* temp;
1409:         temp = new_variable();
1410:         temp->name = yyvsp[0].str;
1411:         yyval.var = temp;
1412: ;}
1413: break;
1414:
1415: case 10:
1416: #line 98 "VarInfoParser.y"
1417: {
1418:         variable* temp;
1419:         temp = new_variable();
1420:         temp->kind = yyvsp[0].str;
1421:         yyval.var = temp;
1422: ;}
1423: break;
1424:
1425: case 11:
1426: #line 105 "VarInfoParser.y"
1427: {
1428:         variable* temp;
1429:         temp = new_variable();
1430:         temp->format = yyvsp[0].str;

```

```

1431:          yyval.var = temp;
1432: ;}
1433:  break;
1434:
1435:  case 12:
1436: #line 112 "VarInfoParser.y"
1437:  {
1438:          variable* temp;
1439:          temp = new_variable();
1440:          temp->qualifier = yyvsp[0].str;
1441:          printf("Qualifier field has been deprecated! \n");
1442:          yyval.var = temp;
1443: ;}
1444:  break;
1445:
1446:  case 13:
1447: #line 120 "VarInfoParser.y"
1448:  {
1449:          variable* temp;
1450:          temp = new_variable();
1451:          temp->id = yyvsp[0].str;
1452:          yyval.var = temp;
1453: ;}
1454:  break;
1455:
1456:  case 14:
1457: #line 127 "VarInfoParser.y"
1458:  {
1459:          variable* temp;
1460:          temp = new_variable();
1461:          temp->description = yyvsp[0].str;
1462:          yyval.var = temp;
1463: ;}
1464:  break;
1465:
1466:  case 15:
1467: #line 134 "VarInfoParser.y"
1468:  {
1469:          variable* temp;
1470:          temp = new_variable();
1471:          temp->range_min = yyvsp[0].str;

```

```

1472:         yyval.var = temp;
1473: ;}
1474:     break;
1475:
1476: case 16:
1477: #line 141 "VarInfoParser.y"
1478:     {
1479:         variable* temp;
1480:         temp = new_variable();
1481:         temp->range_max = yyvsp[0].str;
1482:         yyval.var = temp;
1483: ;}
1484:     break;
1485:
1486: case 17:
1487: #line 148 "VarInfoParser.y"
1488:     {
1489:         variable* temp;
1490:         temp = new_variable();
1491:         temp->length = yyvsp[0].str;
1492:         yyval.var = temp;
1493: ;}
1494:     break;
1495:
1496: case 18:
1497: #line 155 "VarInfoParser.y"
1498:     {
1499:         variable* temp;
1500:         temp = new_variable();
1501:         temp->default_value = yyvsp[0].str;
1502:         yyval.var = temp;
1503: ;}
1504:     break;
1505:
1506: case 19:
1507: #line 162 "VarInfoParser.y"
1508:     {
1509:         variable* temp;
1510:         temp = new_variable();
1511:         temp->precision = yyvsp[0].str;
1512:         yyval.var = temp;

```

```

1513: ;}
1514:  break;
1515:
1516:  case 20:
1517: #line 169 "VarInfoParser.y"
1518:  {
1519:          variable* temp;
1520:          temp = new_variable();
1521:          temp->units = yyvsp[0].str;
1522:          yyval.var = temp;
1523: ;}
1524:  break;
1525:
1526:  case 21:
1527: #line 176 "VarInfoParser.y"
1528:  {
1529:          variable* temp;
1530:          temp = new_variable();
1531:          temp->accuracy = yyvsp[0].str;
1532:          yyval.var = temp;
1533: ;}
1534:  break;
1535:
1536:  case 22:
1537: #line 183 "VarInfoParser.y"
1538:  {
1539:          variable* temp;
1540:          temp = new_variable();
1541:          temp->scale_factor = yyvsp[0].str;
1542:          yyval.var = temp;
1543: ;}
1544:  break;
1545:
1546:  case 23:
1547: #line 190 "VarInfoParser.y"
1548:  {
1549:          variable* temp;
1550:          temp = new_variable();
1551:          temp->scale_units = yyvsp[0].str;
1552:          yyval.var = temp;
1553: ;}

```

```

1554: break;
1555:
1556: case 24:
1557: #line 199 "VarInfoParser.y"
1558: {
1559:         yyval.var = merge_variables(yyvsp[-1].var,yyvsp[0].var);
1560: ;}
1561: break;
1562:
1563: case 25:
1564: #line 205 "VarInfoParser.y"
1565: {
1566:         variable* temp;
1567:         temp = new_variable();
1568:         temp->qualifiers = yyvsp[0].qual;
1569:         yyval.var = temp;
1570: ;}
1571: break;
1572:
1573: case 26:
1574: #line 215 "VarInfoParser.y"
1575: {
1576:         yyval.var = merge_variables(yyvsp[-1].var,yyvsp[0].var);
1577: ;}
1578: break;
1579:
1580: case 27:
1581: #line 219 "VarInfoParser.y"
1582: {
1583:         yyval.var = NULL;
1584: ;}
1585: break;
1586:
1587: case 28:
1588: #line 225 "VarInfoParser.y"
1589: {
1590:         variable* temp;
1591:         temp = new_variable();
1592:         temp->dranges = yyvsp[0].drange;
1593:         yyval.var = temp;
1594: ;}

```

```

1595: break;
1596:
1597: case 29:
1598: #line 232 "VarInfoParser.y"
1599: {
1600:     variable* temp;
1601:     temp = new_variable();
1602:     temp->curves = yyvsp[0].curve;
1603:     yyval.var = temp;
1604: ;}
1605: break;
1606:
1607: case 30:
1608: #line 241 "VarInfoParser.y"
1609: {
1610:     yyval.qual=yyvsp[0].qual
1611: ;}
1612: break;
1613:
1614: case 31:
1615: #line 245 "VarInfoParser.y"
1616: {
1617:     yyval.qual=yyvsp[0].qual
1618: ;}
1619: break;
1620:
1621: case 32:
1622: #line 251 "VarInfoParser.y"
1623: {
1624:     yyval.qual = yyvsp[-4].qual;
1625: ;}
1626: break;
1627:
1628: case 33:
1629: #line 257 "VarInfoParser.y"
1630: {
1631:     yyval.qual = yyvsp[-1].qual;
1632: ;}
1633: break;
1634:
1635: case 35:

```

```

1636: #line 266 "VarInfoParser.y"
1637:  {
1638:          yyval.qual = yyvsp[0].qual;
1639: ;}
1640:  break;
1641:
1642: case 36:
1643: #line 272 "VarInfoParser.y"
1644:  {
1645:          yyval.qual = merge_qualifiers(yyvsp[-1].qual,yyvsp[0].qual);
1646: ;}
1647:  break;
1648:
1649: case 37:
1650: #line 276 "VarInfoParser.y"
1651:  {
1652:          yyval.qual = NULL;
1653: ;}
1654:  break;
1655:
1656: case 38:
1657: #line 282 "VarInfoParser.y"
1658:  {
1659:          qualifier_type* temp;
1660:          temp = new_qualifier();
1661:          temp->name = yyvsp[0].str;
1662:          yyval.qual = temp;
1663: ;}
1664:  break;
1665:
1666: case 39:
1667: #line 289 "VarInfoParser.y"
1668:  {
1669:          qualifier_type* temp;
1670:          temp = new_qualifier();
1671:          temp->value = yyvsp[0].str;
1672:          yyval.qual = temp;
1673: ;}
1674:  break;
1675:
1676: case 40:

```



```

1677: #line 296 "VarInfoParser.y"
1678:  {
1679:      qualifier_type* temp;
1680:      temp = new_qualifier();
1681:      temp->units = yyvsp[0].str;
1682:      yyval.qual = temp;
1683: ;}
1684:  break;
1685:
1686:  case 41:
1687: #line 305 "VarInfoParser.y"
1688:  {
1689:      drange* temp;
1690:      temp = new_drange();
1691:      temp->options = yyvsp[-2].curveoption;
1692:      yyval.drange = merge_dranges(yyvsp[-3].drange,temp);
1693: ;}
1694:  break;
1695:
1696:  case 42:
1697: #line 314 "VarInfoParser.y"
1698:  {
1699:      yyval.drange = yyvsp[-1].drange;
1700: ;}
1701:  break;
1702:
1703:  case 43:
1704: #line 320 "VarInfoParser.y"
1705:  {
1706:      yyval.drange = merge_dranges(yyvsp[-1].drange,yyvsp[0].drange);
1707: ;}
1708:  break;
1709:
1710:  case 44:
1711: #line 324 "VarInfoParser.y"
1712:  {
1713:      yyval.drange = NULL;
1714: ;}
1715:  break;
1716:
1717:  case 45:

```

```

1718: #line 330 "VarInfoParser.y"
1719:  {
1720:      drange* temp;
1721:      temp = new_drange();
1722:      temp->name = yyvsp[0].str;
1723:      yyval.drange = temp;
1724: ;}
1725:  break;
1726:
1727: case 46:
1728: #line 337 "VarInfoParser.y"
1729:  {
1730:      drange* temp;
1731:      temp = new_drange();
1732:      temp->description = yyvsp[0].str;
1733:      yyval.drange = temp;
1734: ;}
1735:  break;
1736:
1737: case 47:
1738: #line 346 "VarInfoParser.y"
1739:  {
1740:      yyval.curveoption = link_options(yyvsp[-1].curveoption,yyvsp[0].curveoption);
1741: ;}
1742:  break;
1743:
1744: case 48:
1745: #line 350 "VarInfoParser.y"
1746:  {
1747:      yyval.curveoption = NULL;
1748: ;}
1749:  break;
1750:
1751: case 49:
1752: #line 356 "VarInfoParser.y"
1753:  {
1754:      yyval.curveoption = yyvsp[-1].curveoption;
1755: ;}
1756:  break;
1757:
1758: case 50:

```

```

1759: #line 360 "VarInfoParser.y"
1760:  {
1761:          yyval.curveoption = yyvsp[-3].curveoption;
1762: ;}
1763:  break;
1764:
1765:  case 51:
1766: #line 366 "VarInfoParser.y"
1767:  {
1768:          yyval.curveoption = merge_options(yyvsp[-1].curveoption,yyvsp[0].curveoption);
1769: ;}
1770:  break;
1771:
1772:  case 52:
1773: #line 370 "VarInfoParser.y"
1774:  {
1775:          yyval.curveoption = NULL;
1776: ;}
1777:  break;
1778:
1779:  case 53:
1780: #line 376 "VarInfoParser.y"
1781:  {
1782:          curveoption* temp;
1783:          temp = new_option();
1784:          temp->name = yyvsp[0].str;
1785:          yyval.curveoption = temp;
1786: ;}
1787:  break;
1788:
1789:  case 54:
1790: #line 383 "VarInfoParser.y"
1791:  {
1792:          curveoption* temp;
1793:          temp = new_option();
1794:          temp->value = yyvsp[0].str;
1795:          yyval.curveoption = temp;
1796: ;}
1797:  break;
1798:
1799:  case 55:

```

```

1800: #line 390 "VarInfoParser.y"
1801:  {
1802:          curveoption* temp;
1803:          temp = new_option();
1804:          temp->description = yyvsp[0].str;
1805:          yyval.curveoption = temp;
1806: ;}
1807:  break;
1808:
1809:  case 56:
1810: #line 397 "VarInfoParser.y"
1811:  {
1812:          curveoption* temp;
1813:          temp = new_option();
1814:          temp->alarm = yyvsp[0].str;
1815:          yyval.curveoption = temp;
1816: ;}
1817:  break;
1818:
1819:  case 57:
1820: #line 405 "VarInfoParser.y"
1821:  {
1822:          curve* temp;
1823:          temp = new_curve();
1824:          temp->coefs = yyvsp[-2].coef;
1825:          yyval.curve = merge_curves(yyvsp[-3].curve,temp);
1826: ;}
1827:  break;
1828:
1829:  case 58:
1830: #line 414 "VarInfoParser.y"
1831:  {
1832:  yyval.curve = yyvsp[-1].curve;
1833: ;}
1834:  break;
1835:
1836:  case 59:
1837: #line 420 "VarInfoParser.y"
1838:  {
1839:  yyval.curve = merge_curves(yyvsp[-1].curve,yyvsp[0].curve);
1840: ;}

```

```

1841:  break;
1842:
1843:  case 60:
1844: #line 424 "VarInfoParser.y"
1845:  {
1846:  yyval.curve = NULL;
1847: ;}
1848:  break;
1849:
1850:  case 61:
1851: #line 430 "VarInfoParser.y"
1852:  {
1853:  curve* temp;
1854:          temp = new_curve();
1855:          temp->name = yyvsp[0].str;
1856:          yyval.curve = temp;
1857: ;}
1858:  break;
1859:
1860:  case 62:
1861: #line 437 "VarInfoParser.y"
1862:  {
1863:  curve* temp;
1864:          temp = new_curve();
1865:          temp->description = yyvsp[0].str;
1866:          yyval.curve = temp;
1867: ;}
1868:  break;
1869:
1870:  case 63:
1871: #line 446 "VarInfoParser.y"
1872:  {
1873:  yyval.coef = link_coefs(yyvsp[-1].coef,yyvsp[0].coef);
1874: ;}
1875:  break;
1876:
1877:  case 64:
1878: #line 450 "VarInfoParser.y"
1879:  {
1880:  yyval.coef = NULL;
1881: ;}

```

```

1882:  break;
1883:
1884:  case 65:
1885: #line 456 "VarInfoParser.y"
1886:  {
1887:  yyval.coef = yyvsp[-1].coef;
1888: ;}
1889:  break;
1890:
1891:  case 66:
1892: #line 460 "VarInfoParser.y"
1893:  {
1894:  yyval.coef = yyvsp[-3].coef;
1895: ;}
1896:  break;
1897:
1898:  case 67:
1899: #line 464 "VarInfoParser.y"
1900:  {
1901:  yyval.coef = yyvsp[-1].coef;
1902: ;}
1903:  break;
1904:
1905:  case 68:
1906: #line 470 "VarInfoParser.y"
1907:  {
1908:  yyval.coef = merge_coefs(yyvsp[-1].coef,yyvsp[0].coef);
1909: ;}
1910:  break;
1911:
1912:  case 69:
1913: #line 474 "VarInfoParser.y"
1914:  {
1915:  yyval.coef = NULL;
1916: ;}
1917:  break;
1918:
1919:  case 70:
1920: #line 480 "VarInfoParser.y"
1921:  {
1922:  coef* temp;

```

```

1923:         temp = new_coef();
1924:         temp->exponent = yyvsp[0].str;
1925:         yyval.coef = temp;
1926: ;}
1927:     break;
1928:
1929: case 71:
1930: #line 487 "VarInfoParser.y"
1931:     {
1932:     coef* temp;
1933:         temp = new_coef();
1934:         temp->value = yyvsp[0].str;
1935:         yyval.coef = temp;
1936: ;}
1937:     break;
1938:
1939: case 72:
1940: #line 494 "VarInfoParser.y"
1941:     {
1942:     coef* temp;
1943:         temp = new_coef();
1944:         temp->description = yyvsp[0].str;
1945:         yyval.coef = temp;
1946: ;}
1947:     break;
1948:
1949: case 73:
1950: #line 502 "VarInfoParser.y"
1951:     {
1952:     yyval.qual= link_qualifiers(yyvsp[-1].qual,yyvsp[0].qual);
1953: ;}
1954:     break;
1955:
1956: case 74:
1957: #line 506 "VarInfoParser.y"
1958:     {
1959:     yyval.qual= NULL;
1960: ;}
1961:     break;
1962:
1963:

```

```

1964:  }
1965:
1966: /* Line 1010 of yacc.c. */
1967: #line 1968 "VarInfoParser.tab.c"
1968:
1969: yyvsp -= yrlen;
1970: yyssp -= yrlen;
1971:
1972:
1973: YY_STACK_PRINT (yyss, yyssp);
1974:
1975: *++yyvsp = yyval;
1976:
1977:
1978: /* Now `shift' the result of the reduction. Determine what state
1979:    that goes to, based on the state we popped back to and the rule
1980:    number reduced by. */
1981:
1982: yyn = yyr1[yyn];
1983:
1984: yystate = yypgoto[yyn - YYNTOKENS] + *yyssp;
1985: if (0 <= yystate && yystate <= YYLAST && yycheck[yystate] == *yyssp)
1986:   yystate = yytable[yystate];
1987: else
1988:   yystate = yydefgoto[yyn - YYNTOKENS];
1989:
1990: goto yynewstate;
1991:
1992:
1993: /*-----
1994: | yyerrlab -- here on detecting error |
1995: `-----*/
1996: yyerrlab:
1997: /* If not already recovering from an error, report this error. */
1998: if (!yyerrstatus)
1999:   {
2000:     ++yynerrs;
2001: #if YYERROR_VERBOSE
2002:     yyn = yypact[yystate];
2003:
2004:     if (YYPACT_NINF < yyn && yyn < YYLAST)

```



```

2005: {
2006:     YYSIZE_T yysize = 0;
2007:     int yytype = YYTRANSLATE (yychar);
2008:     const char* yyprefix;
2009:     char *yymsg;
2010:     int yyx;
2011:
2012:     /* Start YYX at -YYN if negative to avoid negative indexes in
2013:        YYCHECK. */
2014:     int yyxbegin = yyn < 0 ? -yyn : 0;
2015:
2016:     /* Stay within bounds of both yycheck and yytnam. */
2017:     int yychecklim = YYLAST - yyn;
2018:     int yyxend = yychecklim < YYNTOKENS ? yychecklim : YYNTOKENS;
2019:     int yycount = 0;
2020:
2021:     yyprefix = ", expecting ";
2022:     for (yyx = yyxbegin; yyx < yyxend; ++yyx)
2023:         if (yycheck[yyx + yyn] == yyx && yyx != YYTERROR)
2024:             {
2025:                 yysize += yystrlen (yyprefix) + yystrlen (yytnam [yyx]);
2026:                 yycount += 1;
2027:                 if (yycount == 5)
2028:                     {
2029:                         yysize = 0;
2030:                         break;
2031:                     }
2032:             }
2033:     yysize += (sizeof ("syntax error, unexpected ")
2034:                + yystrlen (yytnam[yytype]));
2035:     yymsg = (char *) YYSTACK_ALLOC (ysize);
2036:     if (yymsg != 0)
2037:         {
2038:             char *yyp = yystpcpy (yymsg, "syntax error, unexpected ");
2039:             yyp = yystpcpy (yyp, yytnam[yytype]);
2040:
2041:             if (yycount < 5)
2042:                 {
2043:                     yyprefix = ", expecting ";
2044:                     for (yyx = yyxbegin; yyx < yyxend; ++yyx)
2045:                         if (yycheck[yyx + yyn] == yyx && yyx != YYTERROR)

```

```

2046:     {
2047:         yyp = yystpcpy (yyp, yyprefix);
2048:         yyp = yystpcpy (yyp, yytname[yyx]);
2049:         yyprefix = " or ";
2050:     }
2051: }
2052: yyerror (yymsg);
2053: YYSTACK_FREE (yymsg);
2054: }
2055: else
2056:     yyerror ("syntax error; also virtual memory exhausted");
2057: }
2058: else
2059: #endif /* YYERROR_VERBOSE */
2060: yyerror ("syntax error");
2061: }
2062:
2063:
2064:
2065: if (yyerrstatus == 3)
2066: {
2067:     /* If just tried and failed to reuse lookahead token after an
2068:     error, discard it. */
2069:
2070:     if (yychar <= YYEOF)
2071:     {
2072:         /* If at end of input, pop the error token,
2073:         then the rest of the stack, then return failure. */
2074:         if (yychar == YYEOF)
2075:             for (;;)
2076:             {
2077:                 YYPOPSTACK;
2078:                 if (yyssp == yyss)
2079:                     YYABORT;
2080:                 YYDSYMPRINTF ("Error: popping", yystos[*yyssp], yyvsp, yylsp);
2081:                 yydestruct (yystos[*yyssp], yyvsp);
2082:             }
2083:     }
2084:     else
2085:     {
2086:         YYDSYMPRINTF ("Error: discarding", yytoken, &yylval, &yyllloc);

```

```

2087: yydestruct (yytoken, &yyval);
2088: yychar = YYEMPTY;
2089:
2090: }
2091: }
2092:
2093: /* Else will try to reuse lookahead token after shifting the error
2094:    token. */
2095: goto yyerrlab1;
2096:
2097:
2098: /*-----,
2099: | yyerrorlab -- error raised explicitly by YYERROR. |
2100: `-----*/
2101: yyerrorlab:
2102:
2103: #ifdef __GNUC__
2104: /* Pacify GCC when the user code never invokes YYERROR and the label
2105:    yyerrorlab therefore never appears in user code. */
2106: if (0)
2107:     goto yyerrorlab;
2108: #endif
2109:
2110: yyvsp -= yylen;
2111: yyssp -= yylen;
2112: yystate = *yyssp;
2113: goto yyerrlab1;
2114:
2115:
2116: /*-----,
2117: | yyerrlab1 -- common code for both syntax error and YYERROR. |
2118: `-----*/
2119: yyerrlab1:
2120: yyerrstatus = 3; /* Each real token shifted decrements this. */
2121:
2122: for (;;)
2123: {
2124:     yyn = yypact[yystate];
2125:     if (yyn != YYPACT_NINF)
2126:     {
2127:         yyn += YYTERROR;

```

```

2128:   if (0 <= yyn && yyn <= YYLAST && yyccheck[yyn] == YYTERROR)
2129:   {
2130:       yyn = yytable[yyn];
2131:       if (0 < yyn)
2132:           break;
2133:   }
2134: }
2135:
2136: /* Pop the current state because it cannot handle the error token. */
2137: if (yyssp == yyss)
2138: YYABORT;
2139:
2140: YYDSYMPRINTF ("Error: popping", yystos[*yyssp], yyvsp, yylsp);
2141: yydestruct (yystos[yystate], yyvsp);
2142: YYPOPSTACK;
2143: yystate = *yyssp;
2144: YY_STACK_PRINT (yyss, yyssp);
2145: }
2146:
2147: if (yyn == YYFINAL)
2148: YYACCEPT;
2149:
2150: YYDPRINTF ((stderr, "Shifting error token, "));
2151:
2152: *++yyvsp = yylval;
2153:
2154:
2155: yystate = yyn;
2156: goto yynewstate;
2157:
2158:
2159: /*-----
2160: | yyacceptlab -- YYACCEPT comes here. |
2161: `-----*/
2162: yyacceptlab:
2163: yyresult = 0;
2164: goto yyreturn;
2165:
2166: /*-----
2167: | yyabortlab -- YYABORT comes here. |
2168: `-----*/

```

```

2169: yyabortlab:
2170:  yyresult = 1;
2171:  goto yyreturn;
2172:
2173: #ifndef yyoverflow
2174: /*-----.
2175: | yyoverflowlab -- parser overflow comes here. |
2176: `-----*/
2177: yyoverflowlab:
2178:  yyerror ("parser stack overflow");
2179:  yyresult = 2;
2180:  /* Fall through. */
2181: #endif
2182:
2183: yyreturn:
2184: #ifndef yyoverflow
2185:  if (yyss != yyssa)
2186:    YYSTACK_FREE (yyss);
2187: #endif
2188:  return yyresult;
2189: }
2190:
2191:
2192: #line 511 "VarInfoParser.y"
2193:

```

File: sdm/common/VarInfoParser/VarInfoParser.cpp

```
1: #include <string.h>
2: #include "VarInfoParser.h"
3:
4: VarInfoParser::VarInfoParser():varTree(NULL)
5: {
6: }
7:
8: /*
9:  * Destructor should free the varTree object which is a heap-allocated object, using delete_variable.
10: */
11: VarInfoParser::~VarInfoParser()
12: {
13: if (varTree != NULL)
14: {
15:     delete_variable(varTree);
16:     varTree = NULL;
17: }
18: }
19: /*
20:  * Sets the VarInfo to parse, as returned from a SDMVarInfo message.
21:  * PARAMS:
22:  *     varInfo - The variable information string to parse.
23:  * RETURNS:
24:  *     bool - True upon successful parse, false if a syntax error occurred.
25: */
26: bool VarInfoParser::setVarInfo(char* varInfo)
27: {
28: if (varInfo == NULL) return false;
29: //varTree is a heap-allocated object!
30: if (varTree)
31: {
32:     delete_variable(varTree);
33:     varTree = NULL;
34: }
35: varTree = parseVarInfo(varInfo);
36: if (varTree == NULL)
37:     return false;
38: return true;
39: }
```

```

40:
41: /*****BEGIN GET ATTRIBUTE FUNCTIONS*****/
42:
43: /*
44:  * These functions get the specified attribute of the variable, through the buffer parameter.
45:  * Returns:
46:  *      bool - True if the parameter was set, false otherwise.
47:  */
48: bool VarInfoParser::getLength(char *lengthBuffer)
49: {
50: if (varTree != NULL && varTree->length!=NULL && lengthBuffer!=NULL)
51: {
52:     strcpy(lengthBuffer, varTree->length);
53:     return true;
54: }
55: return false;
56: }
57: bool VarInfoParser::getKind(char *kindBuffer)
58: {
59: if (varTree != NULL && varTree->kind!=NULL && kindBuffer!=NULL)
60: {
61:     strcpy(kindBuffer, varTree->kind);
62:     return true;
63: }
64: return false;
65: }
66: bool VarInfoParser::getName(char *nameBuffer)
67: {
68: if (varTree != NULL && varTree->name!=NULL && nameBuffer!=NULL)
69: {
70:     strcpy(nameBuffer, varTree->name);
71:     return true;
72: }
73: return false;
74: }
75: bool VarInfoParser::getId(char *idBuffer)
76: {
77: if (varTree != NULL && varTree->id!=NULL && idBuffer!=NULL)
78: {
79:     strcpy(idBuffer, varTree->id);
80:     return true;

```

```

81: }
82: return false;
83: }
84: bool VarInfoParser::getRangeMin(char *rminBuffer)
85: {
86: if (varTree != NULL && varTree->range_min!=NULL && rminBuffer!=NULL)
87: {
88:     strcpy(rminBuffer, varTree->range_min);
89:     return true;
90: }
91: return false;
92: }
93: bool VarInfoParser::getRangeMax(char *rmaxBuffer)
94: {
95: if (varTree != NULL && varTree->range_max!=NULL && rmaxBuffer!=NULL)
96: {
97:     strcpy(rmaxBuffer, varTree->range_max);
98:     return true;
99: }
100: return false;
101: }
102: bool VarInfoParser::getDefaultValue(char *dvalueBuffer)
103: {
104: if (varTree != NULL && varTree->default_value!=NULL && dvalueBuffer!=NULL)
105: {
106:     strcpy(dvalueBuffer, varTree->default_value);
107:     return true;
108: }
109: return false;
110: }
111: bool VarInfoParser::getPrecision(char *precisionBuffer)
112: {
113: if (varTree != NULL && varTree->precision!=NULL && precisionBuffer!=NULL)
114: {
115:     strcpy(precisionBuffer, varTree->precision);
116:     return true;
117: }
118: return false;
119: }
120: bool VarInfoParser::getUnits(char *unitsBuffer)
121: {

```



```

122:   if (varTree != NULL && varTree->units!=NULL && unitsBuffer!=NULL)
123:   {
124:       strcpy(unitsBuffer, varTree->units);
125:       return true;
126:   }
127:   return false;
128: }
129: bool VarInfoParser::getAccuracy(char *accBuffer)
130: {
131:   if (varTree != NULL && varTree->accuracy!=NULL && accBuffer!=NULL)
132:   {
133:       strcpy(accBuffer, varTree->accuracy);
134:       return true;
135:   }
136:   return false;
137: }
138: bool VarInfoParser::getScaleFactor(char *sfactorBuffer)
139: {
140:   if (varTree != NULL && varTree->scale_factor!=NULL && sfactorBuffer!=NULL)
141:   {
142:       strcpy(sfactorBuffer, varTree->scale_factor);
143:       return true;
144:   }
145:   return false;
146: }
147: bool VarInfoParser::getScaleUnits(char *sunitsBuffer)
148: {
149:   if (varTree != NULL && varTree->scale_units!=NULL && sunitsBuffer!=NULL)
150:   {
151:       strcpy(sunitsBuffer, varTree->scale_units);
152:       return true;
153:   }
154:   return false;
155: }
156: bool VarInfoParser::getFormat(char *formatBuffer)
157: {
158:   if (varTree != NULL && varTree->format!=NULL && formatBuffer!=NULL)
159:   {
160:       strcpy(formatBuffer, varTree->format);
161:       return true;
162:   }

```

```

163:    return false;
164: }
165: bool VarInfoParser::getDescription(char *descBuffer)
166: {
167:     if (varTree != NULL && varTree->description!=NULL && descBuffer!=NULL)
168:     {
169:         strcpy(descBuffer, varTree->description);
170:         return true;
171:     }
172:     return false;
173: }
174: /*****END GET ATTRIBUTE FUNCTIONS*****/
175:
176:
177: /*
178:  * Finds the value associated with a name for a variable qualifier. The first matching value is
179:  * returned.
180:  * Params:
181:  *     valueBuffer - [OUTPUT] The value associated with the given name, this buffer should be at
182:  *     least 33 bytes.
183:  *     nameBuffer - The name corresponding to the desired value.
184:  * Returns:
185:  *     bool - Returns true if the qualifier was found, false otherwise.
186:  */
187: bool VarInfoParser::getQualValueByName(char *valueBuffer, const char *nameBuffer)
188: {
189:     if (valueBuffer == NULL || nameBuffer == NULL || varTree == NULL) return false;
190:     //Search the qualifier list
191:     for (qualifier_type* Curr = varTree->qualifiers; Curr != NULL; Curr = Curr->next)
192:     {
193:         if (Curr->name == NULL) continue;
194:         if (!strcmp(Curr->name, nameBuffer))
195:         {
196:             strcpy(valueBuffer, Curr->value);
197:             return true;
198:         }
199:     }
200:     return false;
201: }
202: /*
203:  * Determines whether the Variable has qualifiers.

```

```

202: * Returns:
203: *     bool - True if the variables has qualifiers, false otherwise.
204: */
205: bool VarInfoParser::hasQualifiers()
206: {
207:     if (varTree == NULL) return false;
208:     return varTree->qualifiers==NULL ? false : true;
209: }
210: /*
211: * Returns the number of option elements that are associated with the variable's discrete range value
    list.
212: * Returns:
213: *     int - The number of option elements, or zero if one exist.
214: */
215: int VarInfoParser::getDrangeSize()
216: {
217:     if (varTree == NULL || varTree->dranges == NULL) return 0;
218:     int count = 0;
219:     for (curveoption* Curr = varTree->dranges->options; Curr != NULL; Curr = Curr->next,
        count++)
220:         ;
221:     return count;
222: }
223: /*
224: * Gets a name and a value for an option appearing in the position in the drange list defined by the
    optNum argument.
225: * Params:
226: *     optNum - The one-based position counter of the option to get.
227: *     name - A pointer to a buffer for storing the option name.
228: *     value - A pointer to a buffer for storing the value of the option.
229: * Returns:
230: *     bool - True if the optNum is a valid option number that can return values, false otherwise.
231: */
232: bool VarInfoParser::getOptionNumber(int optNum, char *name, char *value)
233: {
234:     if (varTree == NULL || varTree->dranges == NULL || (name == NULL && value == NULL))
        return false;
235:     curveoption* Curr;
236:     int count = 1;
237:     //Find the option
238:     for (Curr = varTree->dranges->options; Curr != NULL && count < optNum; Curr = Curr->next,
        count++)

```

```
239:     ;
240:     //If the optNum was too large to find
241:     if (count != optNum || Curr == NULL)
242:         return false;
243:     if (name != NULL)
244:         strcpy(name, Curr->name);
245:     if (value != NULL)
246:         strcpy(value, Curr->value);
247:     return true;
248:
249: }
250:
```

File: sdm/common/VarInfoParser/VarInfoParser.l

```
1: %{
2: #define _GNU_SOURCE
3: #include <stdio.h>
4: #include <stdlib.h>
5: #include <string.h>
6:
7: #ifndef WIN32
8: #ifndef strndup
9: #define strndup(s,n) __strndup(s,n)
10: #endif
11: #else
12: #include "unistd.h"
13: #endif
14:
15: #include "VarInfoParser.tab.h"
16: %}
17:
18: %%
19:
20: "="                {return EQUAL_SY;}
21: ">"                {return CLOSE_SY;}
22: "/>"              {return SLASHCLOSE_SY;}
23:
24: "<Variable"         {return OPEN_VAR_SY;}
25: "</Variable"        {return CLOSE_VAR_SY;}
26:
27: "<Drange"           {return OPEN_DRANGE_SY;}
28: "</Drange"          {return CLOSE_DRANGE_SY;}
29: "<Option"           {return OPEN_OPTION_SY;}
30: "</Option"          {return CLOSE_OPTION_SY;}
31: "<Qualifier"        {return OPEN_QUALIFIER_SY;}
32: "</Qualifier"       {return CLOSE_QUALIFIER_SY;}
33:
34: "name"              {return NAME_SY;}
35: "kind"              {return KIND_SY;}
36: "id"                {return ID_SY;}
37: "qualifier"         {return QUALIFIER_SY;}
38: "description"       {return DESCRIPTION_SY;}
39: "precision"         {return PRECISION_SY;}
```

```

40: "rangeMax"           {return RANGE_MAX_SY;}
41: "format"             {return FORMAT_SY;}
42: "accuracy"           {return ACCURACY_SY;}
43: "rangeMin"           {return RANGE_MIN_SY;}
44: "scaleFactor"         {return SCALE_FACTOR_SY;}
45: "units"               {return UNITS_SY;}
46: "defaultValue"       {return DEFAULT_VALUE_SY;}
47: "scaleUnits"          {return SCALE_UNITS_SY;}
48: "length"             {return LENGTH_SY;}
49: "value"               {return VALUE_SY;}
50:
51: \"[\"* \"             {VarInfoParserlval.str    =    strdup(yytext+1,strlen(yytext)-2);return
STRING;}
52: [0-9]*[.][0-9]+       {VarInfoParserlval.real = atof(yytext); return FLOAT;}
53: [0-9]+                 {VarInfoParserlval.integer = atoi(yytext); return INT;}
54:
55: "<!--"([^-]*.[^-])*" -->"| "<!-- -->"    /*ignore xteds comments*/
56:
57: [ \\t\\n]*             /*ignore whitespace*/
58: .                       {printf ("Invalid token  \"%s \\n",yytext);}
59:
60: %%
61:
62: int yywrap() {return 1;}
63: void VarInfoParsererror(char *s)
64: {
65: printf("Error occurred at token %s (error is %s) \\n",yytext, s);
66: }

```

File: sdm/common/VarInfoParser/VarInfoParser.h

```
1: #ifndef __VAR_INFO_PARSER_H_
2: #define __VAR_INFO_PARSER_H_
3: extern "C"
4: {
5: #include "Variable.h"
6: #include "../xTEDS/xTEDSParser.h"
7: }
8:
9: /* The VarInfoParser class is a utility class for extracting the variable information data from
SDMVarInfo messages. */
10:
11: class VarInfoParser
12: {
13: public:
14: VarInfoParser();
15: ~VarInfoParser();
16: VarInfoParser(const VarInfoParser &right);
17: VarInfoParser& operator=(const VarInfoParser &right);
18: bool setVarInfo(char* varInfo);           //Used to set the VarInfo return string to parse
19: bool getLength(char *lengthBuffer);       //Used to get the length attribute of the variable
20: bool getKind(char *kindBuffer);           //Used to get the kind attribute of the variable
21: bool getName(char *nameBuffer);           //Used to get the name attribute of the variable
22: bool getId(char *idBuffer);               //Used to get the id attribute of the variable
23: bool getRangeMin(char *rminBuffer);       //Used to get the rangeMin attribute of the variable
24: bool getRangeMax(char *rmaxBuffer);       //Used to get the rangeMax attribute of the variable
25: bool getDefaultValue(char *dvalueBuffer); //Used to get the defaultValue attribute of the variable
26: bool getPrecision(char *precisionBuffer); //Used to get the precision attribute of the variable
27: bool getUnits(char *unitsBuffer);         //Used to get the units attribute of the variable
28: bool getAccuracy(char *accBuffer);        //Used to get the accuracy attribute of the variable
29: bool getScaleFactor(char *sfactorBuffer); //Used to get the scaleFactor attribute of the variable
30: bool getScaleUnits(char *sunitsBuffer);   //Used to get the scaleUnits attribute of the variable
31: bool getFormat(char *formatBuffer);       //Used to get the format attribute of the variable
32: bool getDescription(char *descBuffer);    //Used to get the description attribute of the variable
33: bool getQualValueByName(char *valueBuffer, const char *nameBuffer); //Get qualifier value by
name
34: bool hasQualifiers();                     //Determines whether the variable definition has qualifiers
35: int getDrangeSize();                     //Returns the number of options associated with a Drange
36: bool getOptionNumber(int optNum, char *name, char *value); //Get option element name/value
information
37: private:
```

```
38: variable* varTree;  
39: };  
40:  
41: #endif  
42:
```


File: sdm/common/VarInfoParser/VarInfoParser.tab.h

```
1: /* A Bison parser, made by GNU Bison 1.875d. */
2:
3: /* Skeleton parser for Yacc-like parsing with Bison,
4:  Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.
5:
6:  This program is free software; you can redistribute it and/or modify
7:  it under the terms of the GNU General Public License as published by
8:  the Free Software Foundation; either version 2, or (at your option)
9:  any later version.
10:
11:  This program is distributed in the hope that it will be useful,
12:  but WITHOUT ANY WARRANTY; without even the implied warranty of
13:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14:  GNU General Public License for more details.
15:
16:  You should have received a copy of the GNU General Public License
17:  along with this program; if not, write to the Free Software
18:  Foundation, Inc., 59 Temple Place - Suite 330,
19:  Boston, MA 02111-1307, USA.  */
20:
21: /* As a special exception, when this file is copied by Bison into a
22:  Bison output file, you may use that output file without restriction.
23:  This special exception was added by the Free Software Foundation
24:  in version 1.24 of Bison.  */
25:
26: /* Tokens.  */
27: #ifndef YYTOKENTYPE
28: # define YYTOKENTYPE
29:  /* Put the tokens into the symbol table, so that GDB and other debuggers
30:   know about them.  */
31:  enum yytokentype {
32:    EQUAL_SY = 258,
33:    CLOSE_SY = 259,
34:    SLASHCLOSE_SY = 260,
35:    OPEN_XML_SY = 261,
36:    CLOSE_xTEDS_SY = 262,
37:    OPEN_xTEDS_SY = 263,
38:    OPEN_APP_SY = 264,
39:    OPEN_VAR_SY = 265,
```

40: CLOSE_VAR_SY = 266,
41: OPEN_DRANGE_SY = 267,
42: CLOSE_DRANGE_SY = 268,
43: OPEN_OPTION_SY = 269,
44: OPEN_CURVE_SY = 270,
45: CLOSE_CURVE_SY = 271,
46: OPEN_COEFF_SY = 272,
47: OPEN_DATA_MSG_SY = 273,
48: CLOSE_DATA_MSG_SY = 274,
49: OPEN_VARIABLE_REF_SY = 275,
50: OPEN_COMMAND_MSG_SY = 276,
51: CLOSE_COMMAND_MSG_SY = 277,
52: NAME_SY = 278,
53: KIND_SY = 279,
54: ID_SY = 280,
55: CLOSE_ORIENTATION_SY = 281,
56: QUALIFIER_SY = 282,
57: DESCRIPTION_SY = 283,
58: MANUFACTURER_ID_SY = 284,
59: VERSION_SY = 285,
60: MODEL_ID_SY = 286,
61: VERSION_LETTER_SY = 287,
62: SERIAL_NUMBER_SY = 288,
63: CALIBRATION_DATE_SY = 289,
64: SENSITIVITY_AT_REF_SY = 290,
65: REF_FREQ_SY = 291,
66: REF_TEMP_SY = 292,
67: MEASUREMENT_RANGE_SY = 293,
68: ELECTRICAL_OUTPUT_SY = 294,
69: QUALITY_FACTOR_SY = 295,
70: TEMP_COEFF_SY = 296,
71: DIRECTION_XYZ_SY = 297,
72: CAL_DUE_DATE_SY = 298,
73: POWER_REQS_SY = 299,
74: VALUE_SY = 300,
75: ALARM_SY = 301,
76: MSG_ARRIVAL_SY = 302,
77: MSG_RATE_SY = 303,
78: STRING = 304,
79: FLOAT = 305,
80: INT = 306,

81: PRECISION_SY = 307,
82: RANGE_MAX_SY = 308,
83: CLOSE_LOCATION_SY = 309,
84: FORMAT_SY = 310,
85: ACCURACY_SY = 311,
86: RANGE_MIN_SY = 312,
87: SCALE_FACTOR_SY = 313,
88: UNITS_SY = 314,
89: DEFAULT_VALUE_SY = 315,
90: OPEN_DEVICE_SY = 316,
91: SCALE_UNITS_SY = 317,
92: LENGTH_SY = 318,
93: EXPONENT_SY = 319,
94: SCHEMA_LOCATION_SY = 320,
95: XMLNS_SY = 321,
96: XMLNS_XSI_SY = 322,
97: CLOSE_OPTION_SY = 323,
98: OPEN_INTERFACE_SY = 324,
99: OPEN_COMMAND_SY = 325,
100: OPEN_NOTIFICATION_SY = 326,
101: OPEN_REQUEST_SY = 327,
102: OPEN_FAULT_MSG_SY = 328,
103: COMPONENT_KEY_SY = 329,
104: SPA_U_HUB_SY = 330,
105: SPA_U_PORT_SY = 331,
106: EXTENDS_SY = 332,
107: CLOSE_COMMAND_SY = 333,
108: CLOSE_NOTIFICATION_SY = 334,
109: CLOSE_REQUEST_SY = 335,
110: CLOSE_FAULT_MSG_SY = 336,
111: OPEN_QUALIFIER_SY = 337,
112: CLOSE_QUALIFIER_SY = 338,
113: CLOSE_APP_SY = 339,
114: CLOSE_DEVICE_SY = 340,
115: CLOSE_INTERFACE_SY = 341,
116: MEMORY_MINIMUM_SY = 342,
117: OPERATING_SYSTEM_SY = 343,
118: PATH_FOR_ASSEMBLY_SY = 344,
119: PATH_ON_SPACECRAFT_SY = 345,
120: X_SY = 346,
121: Y_SY = 347,

```

122:  Z_SY = 348,
123:  AXIS_SY = 349,
124:  ANGLE_SY = 350,
125:  OPEN_LOCATION_SY = 351,
126:  OPEN_ORIENTATION_SY = 352,
127:  CLOSE_XML_SY = 353,
128:  ENCODING_SY = 354,
129:  STANDALONE_SY = 355,
130:  CLOSE_VARIABLE_REF_SY = 356,
131:  CLOSE_COEFF_SY = 357
132:  };
133: #endif
134: #define EQUAL_SY 258
135: #define CLOSE_SY 259
136: #define SLASHCLOSE_SY 260
137: #define OPEN_XML_SY 261
138: #define CLOSE_xTEDS_SY 262
139: #define OPEN_xTEDS_SY 263
140: #define OPEN_APP_SY 264
141: #define OPEN_VAR_SY 265
142: #define CLOSE_VAR_SY 266
143: #define OPEN_DRANGE_SY 267
144: #define CLOSE_DRANGE_SY 268
145: #define OPEN_OPTION_SY 269
146: #define OPEN_CURVE_SY 270
147: #define CLOSE_CURVE_SY 271
148: #define OPEN_COEFF_SY 272
149: #define OPEN_DATA_MSG_SY 273
150: #define CLOSE_DATA_MSG_SY 274
151: #define OPEN_VARIABLE_REF_SY 275
152: #define OPEN_COMMAND_MSG_SY 276
153: #define CLOSE_COMMAND_MSG_SY 277
154: #define NAME_SY 278
155: #define KIND_SY 279
156: #define ID_SY 280
157: #define CLOSE_ORIENTATION_SY 281
158: #define QUALIFIER_SY 282
159: #define DESCRIPTION_SY 283
160: #define MANUFACTURER_ID_SY 284
161: #define VERSION_SY 285
162: #define MODEL_ID_SY 286

```

163: #define VERSION_LETTER_SY 287
164: #define SERIAL_NUMBER_SY 288
165: #define CALIBRATION_DATE_SY 289
166: #define SENSITIVITY_AT_REF_SY 290
167: #define REF_FREQ_SY 291
168: #define REF_TEMP_SY 292
169: #define MEASUREMENT_RANGE_SY 293
170: #define ELECTRICAL_OUTPUT_SY 294
171: #define QUALITY_FACTOR_SY 295
172: #define TEMP_COEFF_SY 296
173: #define DIRECTION_XYZ_SY 297
174: #define CAL_DUE_DATE_SY 298
175: #define POWER_REQS_SY 299
176: #define VALUE_SY 300
177: #define ALARM_SY 301
178: #define MSG_ARRIVAL_SY 302
179: #define MSG_RATE_SY 303
180: #define STRING 304
181: #define FLOAT 305
182: #define INT 306
183: #define PRECISION_SY 307
184: #define RANGE_MAX_SY 308
185: #define CLOSE_LOCATION_SY 309
186: #define FORMAT_SY 310
187: #define ACCURACY_SY 311
188: #define RANGE_MIN_SY 312
189: #define SCALE_FACTOR_SY 313
190: #define UNITS_SY 314
191: #define DEFAULT_VALUE_SY 315
192: #define OPEN_DEVICE_SY 316
193: #define SCALE_UNITS_SY 317
194: #define LENGTH_SY 318
195: #define EXPONENT_SY 319
196: #define SCHEMA_LOCATION_SY 320
197: #define XMLNS_SY 321
198: #define XMLNS_XSI_SY 322
199: #define CLOSE_OPTION_SY 323
200: #define OPEN_INTERFACE_SY 324
201: #define OPEN_COMMAND_SY 325
202: #define OPEN_NOTIFICATION_SY 326
203: #define OPEN_REQUEST_SY 327

```

204: #define OPEN_FAULT_MSG_SY 328
205: #define COMPONENT_KEY_SY 329
206: #define SPA_U_HUB_SY 330
207: #define SPA_U_PORT_SY 331
208: #define EXTENDS_SY 332
209: #define CLOSE_COMMAND_SY 333
210: #define CLOSE_NOTIFICATION_SY 334
211: #define CLOSE_REQUEST_SY 335
212: #define CLOSE_FAULT_MSG_SY 336
213: #define OPEN_QUALIFIER_SY 337
214: #define CLOSE_QUALIFIER_SY 338
215: #define CLOSE_APP_SY 339
216: #define CLOSE_DEVICE_SY 340
217: #define CLOSE_INTERFACE_SY 341
218: #define MEMORY_MINIMUM_SY 342
219: #define OPERATING_SYSTEM_SY 343
220: #define PATH_FOR_ASSEMBLY_SY 344
221: #define PATH_ON_SPACECRAFT_SY 345
222: #define X_SY 346
223: #define Y_SY 347
224: #define Z_SY 348
225: #define AXIS_SY 349
226: #define ANGLE_SY 350
227: #define OPEN_LOCATION_SY 351
228: #define OPEN_ORIENTATION_SY 352
229: #define CLOSE_XML_SY 353
230: #define ENCODING_SY 354
231: #define STANDALONE_SY 355
232: #define CLOSE_VARIABLE_REF_SY 356
233: #define CLOSE_COEFF_SY 357
234:
235:
236:
237:
238: #if ! defined (YYSTYPE) && ! defined (YYSTYPE_IS_DECLARED)
239: #line 40 "VarInfoParser.y"
240: typedef union YYSTYPE {
241:     int integer;
242:     float real;
243:     char* str;
244:     struct variable_data* var;

```

```
245: struct qualifier_data* qual;
246: struct coefficient_data* coef;
247: struct curve_data* curve;
248: struct option_data* curveoption;
249: struct drange_data* drange;
250: } YYSTYPE;
251: /* Line 1285 of yacc.c. */
252: #line 253 "VarInfoParser.tab.h"
253: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
254: # define YYSTYPE_IS_DECLARED 1
255: # define YYSTYPE_IS_TRIVIAL 1
256: #endif
257:
258: extern YYSTYPE VarInfoParserlval;
259:
260:
261:
```

File: sdm/common/Exception/SDMException.h

```
1: #ifndef _SDM_EXCEPTION_H_
2: #define _SDM_EXCEPTION_H_
3:
4: #include "../sdmLib.h"
5: #include <string.h>
6:
7: class SDMLIB_API SDMException
8: {
9: public:
10: SDMException();
11: SDMException(const char* ExceptionMessage);
12: virtual ~SDMException();
13:
14: virtual const char* Message() const;
15: private:
16: char m_strMessage[256];
17: };
18:
19:
20: #endif
```


File: sdm/common/Exception/SDMBadIndexException.cpp

```
1: #include "SDMBadIndexException.h"
2:
3: SDMBadIndexException::SDMBadIndexException(const char* strMessage)
4: : SDMException(strMessage)
5: {}
6:
7: SDMBadIndexException::~SDMBadIndexException()
8: {
9: }
10:
11: const char* SDMBadIndexException::Message() const
12: {
13: return SDMException::Message();
14: }
```

File: sdm/common/Exception/Makefile

```
1: include ../../Makefile.common
2: include ../../$(MAKEFILE_DEFS)
3:
4: .PHONY: all clean distclean
5:
6: BUILD_TARGETS=SDMException SDMRegexException SDMBadIndexException
7:
8: all: $(addsuffix .o,$(BUILD_TARGETS))
9:
10: %.o: %.cpp %.h
11: $(CXX) $(CXXFLAGS) -fPIC -c $<
12:
13: clean:
14: rm -f *.o
15:
16: distclean: clean
17: rm -f *~
```

File: sdm/common/Exception/SDMException.cpp

```
1: #include "SDMException.h"
2:
3: SDMException::SDMException()
4: {
5:     m_strMessage[0] = '\0';
6: }
7:
8: SDMException::SDMException(const char* ExceptionMessage)
9: {
10:     strncpy(m_strMessage, ExceptionMessage, sizeof(m_strMessage));
11: }
12:
13: SDMException::~SDMException()
14: {}
15:
16: const char* SDMException::Message() const
17: {
18:     return m_strMessage;
19: }
20:
```

File: sdm/common/Exception/SDMRegexException.h

```
1: #ifndef _SDM_REGEX_EXCEPTION_  
2: #define _SDM_REGEX_EXCEPTION_  
3:  
4: #include "SDMException.h"  
5: #include "../sdmLib.h"  
6:  
7: class SDMLIB_API SDMRegexException : public SDMException  
8: {  
9: public:  
10: SDMRegexException(const char* ExceptionMessage);  
11: virtual ~SDMRegexException();  
12:  
13: virtual const char* Message() const;  
14: };  
15:  
16: #endif  
17:
```

File: sdm/common/Exception/SDMBadIndexException.h

```
1: #ifndef _SDM_BAD_INDEX_EXCEPTION_H_
2: #define _SDM_BAD_INDEX_EXCEPTION_H_
3:
4: #include "SDMException.h"
5: #include "../sdmLib.h"
6:
7: class SDMLIB_API SDMBadIndexException : public SDMException
8: {
9: public:
10: SDMBadIndexException(const char* strMessage);
11: virtual ~SDMBadIndexException();
12: virtual const char* Message() const;
13: };
14:
15: #endif
```

File: sdm/common/Exception/SDMRegexException.cpp

```
1: #include "SDMRegexException.h"
2:
3: SDMRegexException::SDMRegexException(const char* ExceptionMessage)
4:   : SDMException(ExceptionMessage)
5: {}
6:
7: SDMRegexException::~SDMRegexException()
8: {}
9:
10: const char* SDMRegexException::Message() const
11: {
12: return SDMException::Message();
13: }
```

File: sdm/common/asim/asim_win32.h

```
1: #ifndef __ASIM_CLASS_H_
2: #define __ASIM_CLASS_H_
3:
4: #include "asim_commands.h"
5: #include "../asim.h"
6:
7: #include <sys/ioctl.h>
8: #include <stdio.h>
9: typedef void * HANDLE;
10:
11: #include "../sdmLib.h"
12:
13: class SDMLIB_API ASIM
14: {
15: public:
16: ASIM();
17: ASIM(char*);
18: ASIM(const ASIM&);
19: ASIM& operator=(const ASIM&);
20: bool Open(char*);
21: void Close();
22: bool Initialize(void);
23: bool Reset(void);
24: bool SelfTest(void);
25: bool ReqData(unsigned char interface_id,unsigned char msg_id,long ip,short port);
26: bool ReqData(unsigned char interface_id,unsigned char msg_id);
27: bool Cancel(unsigned char interface_id,unsigned char msg_id);
28: bool ReqStream(unsigned char interface_id,unsigned char msg_id,long count);
29: bool PowerOn(void);
30: bool PowerDown(void);
31: bool ReqVersion(void);
32: bool Command(unsigned char interface_id,unsigned char msg_id,short length,unsigned char* data);
33: bool ReqxTEDS(void);
34: bool TimeAtTone(long sec,long usec);
35: char Read(unsigned short& length,unsigned char* buf,int buflen);
36: int RawRead(unsigned char* buf);
37: bool VerifyConnection();
38: char* USBLocation();
39: void SetDebug(int);
```

```
40: private:
41: HANDLE handle;
42: int debug;
43: unsigned char version;
44: char usb_location[80];
45: char devicename[20];
46: FILE* echo;
47: };
48:
49: #endif
50:
```


File: sdm/common/asim/ASIM.cpp

```
1: #include "ASIM.h"
2: #include "../marshall.h"
3: #include "../Time/SDMTime.h"
4:
5: #include <netinet/in.h>
6: #include <string.h>
7: #include <unistd.h>
8: #include <fcntl.h>
9: #include <errno.h>
10: #include <ctype.h>
11:
12: #ifdef __VXWORKS__
13: #include <ioLib.h>
14: #endif
15:
16: #define NO_VERSION 0xFF
17: #define FILE_ECHO 4
18:
19: ASIM::ASIM():handle(-1),debug(0),version(NO_VERSION),echo(NULL)
20: {
21: }
22:
23: ASIM::ASIM(char* devicename):handle(-1),debug(0),version(NO_VERSION),echo(NULL)
24: {
25:     Open(devicename);
26: }
27:
28: ASIM::ASIM(const ASIM& b):handle(b.handle),debug(b.debug),version(b.version),echo(b.echo)
29: {
30: }
31:     strncpy(usb_location,b.usb_location,80);
32:     strncpy(devicename,b.devicename,20);
33: }
34:
35: ASIM::~~ASIM()
36: {
37:     if(handle>0)
38:         close(handle);
39:     handle = -1;
```

```

40:
41: if(debug>=FILE_ECHO)
42: {
43:     fclose(echo);
44: }
45: }
46:
47: ASIM& ASIM::operator=(const ASIM& b)
48: {
49:     handle = b.handle;
50:     debug = b.debug;
51:     version = b.version;
52:     echo = b.echo;
53:     strncpy(usb_location,b.usb_location,80);
54:     strncpy(devicename,b.devicename,20);
55:     return *this;
56: }
57:
58: bool ASIM::Open(char* m_devicename)
59: {
60:     char echo_name[15];
61:     strncpy(devicename,m_devicename,19);
62:     strncpy(echo_name,(m_devicename+5),14);
63:     sprintf(echo_name,"%s.out",echo_name);
64:     if(debug>=FILE_ECHO)
65:     {
66:         echo = fopen(echo_name,"a");
67:     }
68:     handle=open (devicename, O_RDWR);
69:     if (handle < 0)
70:     {
71:         if (errno!=ENOENT)
72:             perror(devicename);
73:         if(debug>=FILE_ECHO)
74:         {
75:             fprintf(echo,"%s: %s \n",devicename,strerror(errno));
76:         }
77:         return false;
78:     }
79:     ioctl(handle, ASIM_TIMEOUT_SEC, 10);
80:     if(debug>=FILE_ECHO)

```

```

81: {
82:     fprintf(echo,"device: %s (%s) \n",devicename,USBLocation());
83:     fflush(echo);
84: }
85: return true;
86: }
87:
88: void ASIM::Close(void)
89: {
90: if(handle>0)
91:     close(handle);
92: handle = -1;
93:
94: if(debug>=FILE_ECHO)
95: {
96:     fclose(echo);
97: }
98: }
99:
100: bool ASIM::Initialize(void)
101: {
102:     char msg[8];
103:     long result = 0;
104:     memset(msg,0,8);
105:     msg[0] = ASIM_INITIALIZE;
106:     msg[1] = msg[2] = 0;
107:     result = write(handle,msg,3);
108:     if(debug>=FILE_ECHO)
109:     {
110:         fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
111:         for(int i=0;i<3;i++)
112:             fprintf(echo,"%hhx ",msg[i]);
113:         fprintf(echo," \n");
114:         fflush(echo);
115:     }
116:     if (result < 3) return false;
117:     return true;
118: }
119: bool ASIM::Reset(void)
120: {
121:     char msg[8];

```

```

122: long result = 0;
123: memset(msg,0,8);
124: msg[0] = ASIM_RESET;
125: msg[1] = msg[2] = 0;
126: result =write(handle,msg,3);
127: if(debug>=FILE_ECHO)
128: {
129:     fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
130:     for(int i=0;i<3;i++)
131:         fprintf(echo,"%hhx ",msg[i]);
132:     fprintf(echo," \n");
133:     fflush(echo);
134: }
135: if (result < 3) return false;
136: return true;
137: }
138: bool ASIM::SelfTest(void)
139: {
140:     char msg[8];
141:     long result = 0;
142:     memset(msg,0,8);
143:     msg[0] = ASIM_SELF_TEST;
144:     msg[1] = msg[2] = 0;
145:     result =write(handle,msg,3);
146:     if(debug>=FILE_ECHO)
147:     {
148:         fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
149:         for(int i=0;i<3;i++)
150:             fprintf(echo,"%hhx ",msg[i]);
151:         fprintf(echo," \n");
152:         fflush(echo);
153:     }
154:     if (result < 3) return false;
155:     return true;
156: }
157:
158: bool ASIM::ReqData(unsigned char interface_id,unsigned char msg_id,long ip, short port)
159: {
160:     unsigned char msg[16];
161:     short length = 8;
162:     long result = 0;

```

```

163:
164:   if (handle < 0)
165:       return false;
166:   memset(msg,0,16);
167:   msg[0] = ASIM_REQ_DATA;
168:   length = htons(length); //length needs to be big-endian
169:   memcpy(&msg[1], &length, 2);
170:   PUT_UCHAR(&msg[3], interface_id);
171:   PUT_UCHAR(&msg[4],msg_id);
172:   PUT_LONG(&msg[5],ip);
173:   PUT_SHORT(&msg[9],port);
174:   result =write(handle,msg,11);
175:   if(debug>=FILE_ECHO)
176:   {
177:       fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
178:       for(int i=0;i<11;i++)
179:           fprintf(echo,"%hhx ",msg[i]);
180:       fprintf(echo," \n");
181:       fflush(echo);
182:   }
183:   if (result < 4) return false;
184:   return true;
185: }
186:
187: bool ASIM::ReqData(unsigned char interface_id,unsigned char msg_id)
188: {
189:     return ReqData(interface_id,msg_id,0,0);
190: }
191:
192: bool ASIM::Cancel(unsigned char interface_id,unsigned char msg_id)
193: {
194:     unsigned char msg[8];
195:     short length = 2;
196:     long result = 0;
197:     memset(msg,0,8);
198:     msg[0] = ASIM_CANCEL;
199:     length = htons(length); //length needs to be big-endian
200:     memcpy(msg+1,&length,2);
201:     msg[3] = interface_id;
202:     msg[4] = msg_id;
203:     result =write(handle,msg,5);

```

```

204:   if(debug>=FILE_ECHO)
205:   {
206:       fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
207:       for(int i=0;i<5;i++)
208:           fprintf(echo,"%hhx ",msg[i]);
209:       fprintf(echo," \n");
210:       fflush(echo);
211:   }
212:   if (result < 4) return false;
213:   return true;
214: }
215: bool ASIM::ReqStream(unsigned char interface_id,unsigned char msg_id,long count)
216: {
217:     unsigned char msg[16];
218:     short length = 6;
219:     long le_count;           //little endian count
220:     long result = 0;
221:     memset(msg,0,16);
222:     msg[0] = ASIM_REQ_STREAM;
223:     length = htons(length);   //length needs to be big-endian
224:     memcpy(msg+1,&length,2);
225:     msg[3] = interface_id;
226:     msg[4] = msg_id;
227:     le_count = SDM_htonl(count); //count needs to be little-endian
228:     memcpy(msg+5,&le_count,4);
229:     result =write(handle,msg,9);
230:     if(debug>=FILE_ECHO)
231:     {
232:         fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
233:         for(int i=0;i<9;i++)
234:             fprintf(echo,"%hhx ",msg[i]);
235:         fprintf(echo," \n");
236:         fflush(echo);
237:     }
238:     if (result < 9) return false;
239:     return true;
240: }
241: bool ASIM::PowerOn(void)
242: {
243:     char msg[8];
244:     long result = 0;

```

```

245:  memset(msg,0,8);
246:  msg[0] = ASIM_POWER_ON;
247:  result = write(handle,msg,3);
248:  if(debug>=FILE_ECHO)
249:  {
250:      fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
251:      for(int i=0;i<3;i++)
252:          fprintf(echo,"%hhx ",msg[i]);
253:      fprintf(echo," \n");
254:      fflush(echo);
255:  }
256:  if (result < 3) return false;
257:  return true;
258: }
259: bool ASIM::PowerDown(void)
260: {
261:     char msg[8];
262:     long result = 0;
263:     memset(msg,0,8);
264:     msg[0] = ASIM_POWER_DOWN;
265:     msg[1] = msg[2] = 0;
266:     result = write(handle,msg,3);
267:     if(debug>=FILE_ECHO)
268:     {
269:         fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
270:         for(int i=0;i<3;i++)
271:             fprintf(echo,"%hhx ",msg[i]);
272:         fprintf(echo," \n");
273:         fflush(echo);
274:     }
275:     if (result < 3) return false;
276:     return true;
277: }
278: bool ASIM::ReqVersion(void)
279: {
280:     char msg[8];
281:     long result = 0;
282:     memset(msg,0,8);
283:     msg[0] = ASIM_REQ_VERSION;
284:     msg[1] = msg[2] = 0;
285:     result = write(handle,msg,3);

```

```

286:  if(debug>=FILE_ECHO)
287:  {
288:      fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
289:      for(int i=0;i<3;i++)
290:          fprintf(echo,"%hhx ",msg[i]);
291:      fprintf(echo," \n");
292:      fflush(echo);
293:  }
294:  if (result < 3) return false;
295:  return true;
296: }
297: bool ASIM::Command(unsigned char interface_id,unsigned char msg_id,short length,unsigned
char* data)
298: {
299:     unsigned char msg[64];
300:     const unsigned short DataOffset = 5;
301:     short be_length;//length needs to be big endian in message
302:     long result = 0;
303:     if (static_cast<unsigned int>(length + DataOffset) > sizeof(msg) || length < 0)
304:         return false;
305:     memset(msg,0,64);
306:     msg[0] = ASIM_COMMAND;
307:     be_length = length + 2; //add two bytes for interface_id and msg_id
308:     be_length = htons(be_length);
309:     memcpy(msg+1,&be_length,2);
310:     msg[3] = interface_id;
311:     msg[4] = msg_id;
312:     memcpy(msg+5,data,length);
313:     result =write(handle,msg,5+length);
314:     if(debug>=FILE_ECHO)
315:     {
316:         fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
317:         for(int i=0;i<5+length;i++)
318:             fprintf(echo,"%hhx ",msg[i]);
319:         fprintf(echo," \n");
320:         fflush(echo);
321:     }
322:     if (result < length+5) return false;
323:     return true;
324: }
325:

```



```

326: bool ASIM::ReqxTEDS(void)
327: {
328:     char msg[8];
329:     long result = 0;
330:     memset(msg,0,8);
331:     msg[0] = ASIM_REQ_XTEDS;
332:     msg[1] = msg[2] = 0;
333:     result = write(handle,msg,3);
334:     if(debug >= FILE_ECHO)
335:     {
336:         fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
337:         for(int i=0;i<3;i++)
338:             fprintf(echo,"%hhx ",msg[i]);
339:         fprintf(echo," \n");
340:         fflush(echo);
341:     }
342:     if (result < 3) return false;
343:     return true;
344: }
345:
346: bool ASIM::TimeAtTone(long sec,long usec)
347: {
348:     unsigned char msg[16];
349:     short length = 8;
350:     long result = 0;
351:     memset(msg,0,16);
352:     msg[0] = ASIM_TIME_AT_TONE;
353:     length = htons(length); //length needs to be big-endian
354:     PUT_SHORT(&msg[1],length);
355:     PUT_LONG(&msg[3],sec);
356:     PUT_LONG(&msg[7],usec);
357:     result = write(handle,msg,11);
358:     if(debug >= FILE_ECHO)
359:     {
360:         fprintf(echo,"message sent (%c,%ld): ",msg[0],result);
361:         for(int i=0;i<11;i++)
362:             fprintf(echo,"%hhx ",msg[i]);
363:         fprintf(echo," \n");
364:         fflush(echo);
365:     }
366:     if (result < 11) return false;

```

```

367:     return true;
368: }
369:
370: int ASIM::RawRead(unsigned char* data)
371: {
372:     unsigned char msg[ASIM_MAX_OUT]; //buffer for message
373:     long result = 0;                  //result of a read
374:     if (handle < 0)
375:     {
376:         if(debug >= FILE_ECHO)
377:         {
378:             fprintf(echo, "ERROR: Bad handle \n");
379:         }
380:         return ASIM_ERROR;
381:     }
382:     result = read(handle, msg, ASIM_MAX_OUT); //read first data message
383:     if (result < 0)
384:     {
385:         if(debug >= FILE_ECHO)
386:         {
387:             fprintf(echo, "ERROR: %s \n", strerror(errno));
388:         }
389:         if(errno == 110)
390:             return -1;
391:         return -1;
392:     }
393:     memcpy(data, msg, result);
394:     return result;
395: }
396:
397: char ASIM::Read(unsigned short& length, unsigned char* data, unsigned int buflen)
398: {
399:     unsigned char msg[ASIM_MAX_OUT]; //buffer for message
400:     unsigned char msg_type;           //first character of message
401:     unsigned char header_buf[3];      //save the header for debug output
402:     long result = 0;                  //result of a read
403:     int bytes_so_far;                 //number of bytes read so far
404:     int bytes_received = 0;
405:     length = 0;                       //if an error occurs before the length field
406:                                         //can be read return 0 as the length
407:

```

```

408:  memset(msg,0,ASIM_MAX_OUT);
409:  if (handle < 0)
410:  {
411:      if(debug>=FILE_ECHO)
412:      {
413:          fprintf(echo,"ERROR: Bad handle \n");
414:      }
415:      return ASIM_ERROR;
416:  }
417:  // Be sure that we at least get a header
418:  while (bytes_received < 3)
419:  {
420:      result = read(handle,msg+bytes_received,ASIM_MAX_OUT-bytes_received);  //read first
data message
421:      if (result <= 0)
422:      {
423:          if(debug>=FILE_ECHO)
424:          {
425:              fprintf(echo,"ERROR: %s \n",strerror(errno));
426:          }
427:          //Some version of the ASIM driver will return zero bytes
428:          if(result == 0 || errno==110)
429:              return ASIM_TIMEOUT;
430:          return ASIM_ERROR;
431:      }
432:      else
433:          bytes_received += result;
434:  }
435:  msg_type = msg[0];          //save message type
436:  // Save the header for debug output
437:  for (unsigned int i = 0; i < sizeof(header_buf); i++)
438:      header_buf[i] = msg[i];
439:
440:  if (msg_type == ASIM_VERSION)    //version needed to know endianness of length
441:  {
442:      //set version
443:      version = msg[3];
444:  }
445:
446:  //get length based on version
447:  if(version != NO_VERSION)

```

```

448:  {
449:      length = msg[2] + (msg[1] << 8);    //length is big endian, convert to host byte order
450:  }
451:  else
452:  {
453:      length = msg[1] + (msg[2] << 8);    //length is little endian, convert to host byte order
454:  }
455:  bytes_so_far = bytes_received - 3;      //bytes read is the total number - header
456:
457:  if (bytes_received >= 3)
458:      memcpy(data,msg+3,bytes_received-3);    //store data bytes
459:
460:  if (msg_type == ASIM_STATUS)            //status always has length 1
461:  {
462:      //but may occur before a version
463:      length = 1;                        //possibly giving a wrong value of 256
464:  }
465:  if (length > buflen)
466:  {
467:      if (debug >= FILE_ECHO)
468:          fprintf(echo, "ERROR: Produced message with invalid length %d bytes.",length);
469:      return ASIM_ERROR;
470:  }
471:  while(length > bytes_so_far)            //continue reading data
472:  {
473:      result = read(handle,msg,ASIM_MAX_OUT);
474:      if (result <= 0)
475:      {
476:          if(debug>=FILE_ECHO)
477:          {
478:              fprintf(echo,"ERROR: %s \n",strerror(errno));
479:          }
480:          if(errno==110)
481:              return ASIM_TIMEOUT;
482:          //Some versions of the ASIM driver will return zero bytes
483:          if (result == 0)
484:          {
485:              usleep(100);
486:              continue;
487:          }
488:          return ASIM_ERROR;
489:      }

```

```

489:     memcpy(data+bytes_so_far,msg,result); //store data bytes
490:     bytes_so_far+=result;
491: }
492:
493: if(debug>=FILE_ECHO)
494: {
495:     // Output header
496:     fprintf(echo,"message rec'd (%c,%hu): %hhx %hhx %hhx ",header_buf[0], length+3,
header_buf[0], header_buf[1], header_buf[2]);
497:     // Output body
498:     for(int i=0;i<length;i++)
499:     {
500:         if ((msg_type == ASIM_XTEDS || msg_type == ASIM_XTEDS_ID_PAIR) &&
(isprint(data[i]) || isspace(data[i])))
501:             fprintf(echo,"%c",data[i]);
502:         else
503:             fprintf(echo," %hhx",data[i]);
504:     }
505:     fprintf(echo," \n");
506:     fflush(echo);
507: }
508: return msg_type;
509: }
510:
511: bool ASIM::VerifyConnection(void)
512: {
513:     int status,product;
514:     long verification_handle;
515:     verification_handle = open (devicename, O_RDWR);
516:     if (verification_handle < 0)
517:     {
518:         close(verification_handle);
519:         return false;
520:     }
521:     status = ioctl (verification_handle, ASIM_PRODUCT_ID, &product);
522:     close(verification_handle);
523:     if (verification_handle < 0) return false;
524:     //if (product == 2) return false;
525:     return true;
526: }
527:

```

```

528: char* ASIM::USBLocation(void)
529: {
530:     char temp_usb[80];
531:     int start_usb_location;
532:     long result = 0;
533:
534:     //initialize
535:     memset(temp_usb,0,80);
536:     memset(usb_location,0,80);
537:     //request USB path
538:     result = ioctl(handle, ASIM_PATH, temp_usb);
539:     //strip unnecessary info
540:     for(start_usb_location = 0;start_usb_location<80;++start_usb_location)
541:         if (temp_usb[start_usb_location]==' ') break;
542:     ++start_usb_location;
543:     //copy to usb member variable
544:     strncpy(usb_location,temp_usb+start_usb_location,79);
545:     return usb_location;
546: }
547:
548: void ASIM::SetDebug(int d)
549: {
550:     debug = d;
551: }
552:
553:

```

File: sdm/common/asim/ASIM.h

```
1: #ifndef __ASIM_CLASS_H_
2: #define __ASIM_CLASS_H_
3:
4: #include "asim_commands.h"
5: #include "../asim.h"
6:
7: #include <sys/ioctl.h>
8: #include <stdio.h>
9:
10:
11: class ASIM
12: {
13: public:
14: ASIM();
15: ASIM(char*);
16: ASIM(const ASIM&);
17: ~ASIM();
18: ASIM& operator=(const ASIM&);
19: bool Open(char*);
20: void Close();
21: bool Initialize(void);
22: bool Reset(void);
23: bool SelfTest(void);
24: bool ReqData(unsigned char interface_id,unsigned char msg_id,long ip,short port);
25: bool ReqData(unsigned char interface_id,unsigned char msg_id);
26: bool Cancel(unsigned char interface_id,unsigned char msg_id);
27: bool ReqStream(unsigned char interface_id,unsigned char msg_id,long count);
28: bool PowerOn(void);
29: bool PowerDown(void);
30: bool ReqVersion(void);
31: bool Command(unsigned char interface_id,unsigned char msg_id,short length,unsigned char* data);
32: bool ReqxTEDS(void);
33: bool TimeAtTone(long sec,long usec);
34: char Read(unsigned short& length,unsigned char* buf, unsigned int buflen);
35: int RawRead(unsigned char* buf);
36: bool VerifyConnection();
37: char* USBLocation();
38: void SetDebug(int);
39: private:
```

```
40: long handle;
41: int debug;
42: unsigned char version;
43: char usb_location[80];
44: char devicename[20];
45: FILE* echo;
46: };
47:
48: #endif
49:
```


File: sdm/common/asim/Makefile

```
1: #asim makefile
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: .PHONY:    all clean distclean
7:
8: all:   ASIM.o
9:
10: ASIM.o:   ASIM.cpp ASIM.h asim_commands.h
11: $(CXX) $(CXXFLAGS) -fPIC -c $<
12:
13: clean:
14: rm -f *.o *~
15:
16: distclean:  clean
```

File: sdm/common/asim/asim_win32.cpp

```
1: #include "asim_win32.h"
2: #include "../marshall.h"
3:
4: #include <netinet/in.h>
5: #include <string.h>
6: #include <unistd.h>
7: #include <fcntl.h>
8: #include <errno.h>
9: #include <windows.h>
10: #define NO_VERSION 0xFF
11: #define FILE_ECHO 4
12:
13: #define ASIM_IN_PIPE 0
14: #define ASIM_OUT_PIPE 1
15: #define ASIM_IOCTL_INDEX 0x0800
16: #define FILE_DEVICE_UNKNOWN 0x00000022
17: #define METHOD_IN_DIRECT 1
18: #define METHOD_OUT_DIRECT 2
19: #define FILE_ANY_ACCESS 0
20: #define CTL_CODE( DeviceType, Function, Method, Access ) ( \
21:  ((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method) \
22: )
23: // Perform a bulk read from the pipe specified in the first byte of the input buffer
24: // The length is interpreted from the buffer lengths specified in the call to DeviceIOControl
25: #define IOCTL_ASIM_BULK_READ CTL_CODE(FILE_DEVICE_UNKNOWN, \
26: ASIM_IOCTL_INDEX+3, \
27: METHOD_OUT_DIRECT, \
28: FILE_ANY_ACCESS)
29:
30: // Perform a bulk write to the pipe specified in the first byte of the input buffer
31: // The length is interpreted from the buffer lengths specified in the call to DeviceIOControl
32: #define IOCTL_ASIM_BULK_WRITE CTL_CODE(FILE_DEVICE_UNKNOWN, \
33: ASIM_IOCTL_INDEX+4, \
34: METHOD_IN_DIRECT, \
35: FILE_ANY_ACCESS)
36:
37: ASIM::ASIM():handle(NULL),debug(0),version(NO_VERSION),echo(NULL)
```

```

38: {
39: }
40:
41: ASIM::ASIM(char* devicename):handle(NULL),debug(0),version(NO_VERSION),echo(NULL)
42: {
43:   Open(devicename);
44: }
45:
46: ASIM::ASIM(const ASIM& b):handle(b.handle),debug(b.debug),version(b.version),echo(b.echo)
47: {
48: {
49:   strncpy(usb_location,b.usb_location,80);
50:   strncpy(devicename,b.devicename,20);
51: };
52:
53: ASIM& ASIM::operator=(const ASIM& b)
54: {
55:   handle = b.handle;
56:   debug = b.debug;
57:   version = b.version;
58:   echo = b.echo;
59:   strncpy(usb_location,b.usb_location,80);
60:   strncpy(devicename,b.devicename,20);
61:   return *this;
62: }
63:
64: bool ASIM::Open(char* m_devicename)
65: {
66:   char echo_name[15];
67:   strncpy(devicename,m_devicename,19);
68:   strncpy(echo_name,(m_devicename+5),14);
69:   sprintf(echo_name,"%s.out",echo_name);
70:   if(debug>=FILE_ECHO)
71:   {
72:     echo = fopen(echo_name,"a");
73:   }
74:   handle = CreateFile(devicename, GENERIC_READ|GENERIC_WRITE,
FILE_SHARE_READ|FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
75:   if (handle == INVALID_HANDLE_VALUE)
76:   {
77:     if (debug >= FILE_ECHO)

```

```

78:     {
79:         fprintf(echo,"%s: %s \n",devicename,strerror(errno));
80:     }
81:     return false;
82: }
83: if(debug>=FILE_ECHO)
84: {
85:     fprintf(echo,"device: %s (%s) \n",devicename,USBLocation());
86:     fflush(echo);
87: }
88: return true;
89: }
90:
91: void ASIM::Close(void)
92: {
93: if (handle != NULL)
94:     CloseHandle(handle);
95: handle = NULL;
96: if(debug>=FILE_ECHO)
97: {
98:     fclose(echo);
99: }
100: }
101:
102: bool ASIM::Initialize(void)
103: {
104:     char msg[8];
105:     unsigned long result = 0;
106:     memset(msg,0,8);
107:     msg[0] = ASIM_INITIALIZE;
108:     msg[1] = msg[2] = 0;
109:     unsigned char pipenum = ASIM_OUT_PIPE;
110:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
111: 3, &result, NULL))
112:         return false;
113:     if(debug>=FILE_ECHO)
114:     {
115:         fprintf(echo,"message sent(%ld): ",result);
116:         for(int i=0;i<3;i++)
117:             fprintf(echo,"%hhx ",msg[i]);
118:         fprintf(echo," \n");

```

```

118:     fflush(echo);
119: }
120: if (result < 3) return false;
121: return true;
122: }
123: bool ASIM::Reset(void)
124: {
125:     char msg[8];
126:     unsigned long result = 0;
127:     memset(msg,0,8);
128:     msg[0] = ASIM_RESET;
129:     msg[1] = msg[2] = 0;
130:     unsigned char pipenum = ASIM_OUT_PIPE;
131:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
132: 3, &result, NULL))
132:         return false;
133:     if(debug>=FILE_ECHO)
134:     {
135:         fprintf(echo,"message sent(%ld): ",result);
136:         for(int i=0;i<3;i++)
137:             fprintf(echo,"%hhx ",msg[i]);
138:         fprintf(echo," \n");
139:         fflush(echo);
140:     }
141:     if (result < 3) return false;
142:     return true;
143: }
144: bool ASIM::SelfTest(void)
145: {
146:     char msg[8];
147:     unsigned long result = 0;
148:     memset(msg,0,8);
149:     msg[0] = ASIM_SELF_TEST;
150:     msg[1] = msg[2] = 0;
151:     unsigned char pipenum = ASIM_OUT_PIPE;
152:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
153: 3, &result, NULL))
153:         return false;
154:     if(debug>=FILE_ECHO)
155:     {
156:         fprintf(echo,"message sent(%ld): ",result);

```

```

157:     for(int i=0;i<3;i++)
158:         fprintf(echo,"%hhx ",msg[i]);
159:     fprintf(echo," \n");
160:     fflush(echo);
161: }
162: if (result < 3) return false;
163: return true;
164: }
165:
166: bool ASIM::ReqData(unsigned char interface_id,unsigned char msg_id,long ip, short port)
167: {
168:     unsigned char msg[16];
169:     short length = 8;
170:     unsigned long result = 0;
171:
172:     if (handle < 0)
173:         return false;
174:     memset(msg,0,16);
175:     msg[0] = ASIM_REQ_DATA;
176:     length = htons(length); //length needs to be big-endian
177:     memcpy(&msg[1], &length, 2);
178:     PUT_UCHAR(&msg[3], interface_id);
179:     PUT_UCHAR(&msg[4],msg_id);
180:     PUT_LONG(&msg[5],ip);
181:     PUT_SHORT(&msg[9],port);
182:     unsigned char pipenum = ASIM_OUT_PIPE;
183:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
184: 11, &result, NULL))
185:         return false;
186:     if(debug>=FILE_ECHO)
187:     {
188:         fprintf(echo,"message sent(%ld): ",result);
189:         for(int i=0;i<11;i++)
190:             fprintf(echo,"%hhx ",msg[i]);
191:         fprintf(echo," \n");
192:         fflush(echo);
193:     }
194:     if (result < 4) return false;
195:     return true;
196: }

```

```

197: bool ASIM::ReqData(unsigned char interface_id,unsigned char msg_id)
198: {
199:     return ReqData(interface_id,msg_id,0,0);
200: }
201:
202: bool ASIM::Cancel(unsigned char interface_id,unsigned char msg_id)
203: {
204:     unsigned char msg[8];
205:     short length = 2;
206:     unsigned long result = 0;
207:     memset(msg,0,8);
208:     msg[0] = ASIM_CANCEL;
209:     length = htons(length); //length needs to be big-endian
210:     memcpy(msg+1,&length,2);
211:     msg[3] = interface_id;
212:     msg[4] = msg_id;
213:     unsigned char pipenum = ASIM_OUT_PIPE;
214:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
215: 5, &result, NULL))
216:         return false;
217:     if(debug>=FILE_ECHO)
218:     {
219:         fprintf(echo,"message sent(%ld): ",result);
220:         for(int i=0;i<5;i++)
221:             fprintf(echo,"%hhx ",msg[i]);
222:         fprintf(echo," \n");
223:         fflush(echo);
224:     }
225:     if (result < 4) return false;
226:     return true;
227: }
228:
229: bool ASIM::ReqStream(unsigned char interface_id,unsigned char msg_id,long count)
230: {
231:     unsigned char msg[16];
232:     short length = 6;
233:     long le_count;           //little endian count
234:     unsigned long result = 0;
235:     memset(msg,0,16);
236:     msg[0] = ASIM_REQ_STREAM;
237:     length = htons(length); //length needs to be big-endian
238:     memcpy(msg+1,&length,2);

```

```

237:  msg[3] = interface_id;
238:  msg[4] = msg_id;
239:  le_count = SDM_htonl(count); //count needs to be little-endian
240:  memcpy(msg+5,&le_count,4);
241:  unsigned char pipenum = ASIM_OUT_PIPE;
242:  if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
243:  9, &result, NULL))
244:      return false;
245:  if(debug>=FILE_ECHO)
246:  {
247:      fprintf(echo,"message sent(%ld): ",result);
248:      for(int i=0;i<9;i++)
249:          fprintf(echo,"%hhx ",msg[i]);
250:      fprintf(echo," \n");
251:      fflush(echo);
252:  }
253:  if (result < 9) return false;
254:  return true;
255: }
256: bool ASIM::PowerOn(void)
257: {
258:     char msg[8];
259:     unsigned long result = 0;
260:     memset(msg,0,8);
261:     msg[0] = ASIM_POWER_ON;
262:     unsigned char pipenum = ASIM_OUT_PIPE;
263:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
264:     3, &result, NULL))
265:         return false;
266:     if(debug>=FILE_ECHO)
267:     {
268:         fprintf(echo,"message sent(%ld): ",result);
269:         for(int i=0;i<3;i++)
270:             fprintf(echo,"%hhx ",msg[i]);
271:         fprintf(echo," \n");
272:         fflush(echo);
273:     }
274:     if (result < 3) return false;
275:     return true;
276: }
277: bool ASIM::PowerDown(void)

```



```

276: {
277:     char msg[8];
278:     unsigned long result = 0;
279:     memset(msg,0,8);
280:     msg[0] = ASIM_POWER_DOWN;
281:     msg[1] = msg[2] = 0;
282:     unsigned char pipenum = ASIM_OUT_PIPE;
283:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
284: 3, &result, NULL))
285:         return false;
286:     if(debug>=FILE_ECHO)
287:     {
288:         fprintf(echo,"message sent(%ld): ",result);
289:         for(int i=0;i<3;i++)
290:             fprintf(echo,"%hhx ",msg[i]);
291:         fprintf(echo," \n");
292:         fflush(echo);
293:     }
294:     if (result < 3) return false;
295:     return true;
296: }
297: bool ASIM::ReqVersion(void)
298: {
299:     char msg[8];
300:     unsigned long result = 0;
301:     memset(msg,0,8);
302:     msg[0] = ASIM_REQ_VERSION;
303:     msg[1] = msg[2] = 0;
304:     unsigned char pipenum = ASIM_OUT_PIPE;
305:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
306: 3, &result, NULL))
307:         return false;
308:     if(debug>=FILE_ECHO)
309:     {
310:         fprintf(echo,"message sent(%ld): ",result);
311:         for(int i=0;i<3;i++)
312:             fprintf(echo,"%hhx ",msg[i]);
313:         fprintf(echo," \n");
314:         fflush(echo);
315:     }
316:     if (result < 3) return false;

```

```

315:     return true;
316: }
317: bool ASIM::Command(unsigned char interface_id,unsigned char msg_id,short length,unsigned
char* data)
318: {
319:     unsigned char msg[64];
320:     short be_length;//length needs to be big endian in message
321:     unsigned long result = 0;
322:     memset(msg,0,64);
323:     msg[0] = ASIM_COMMAND;
324:     be_length = length + 2; //add two bytes for interface_id and msg_id
325:     be_length = htons(be_length);
326:     memcpy(msg+1,&be_length,2);
327:     msg[3] = interface_id;
328:     msg[4] = msg_id;
329:     memcpy(msg+5,data,length);
330:     unsigned char pipenum = ASIM_OUT_PIPE;
331:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
5+length, &result, NULL))
332:         return false;
333:     if(debug>=FILE_ECHO)
334:     {
335:         fprintf(echo,"message sent(%ld): ",result);
336:         for(int i=0;i<5+length;i++)
337:             fprintf(echo,"%hhx ",msg[i]);
338:         fprintf(echo," \n");
339:         fflush(echo);
340:     }
341:     if (result < length+5) return false;
342:     return true;
343: }
344:
345: bool ASIM::ReqxTEDS(void)
346: {
347:     char msg[8];
348:     unsigned long result = 0;
349:     memset(msg,0,8);
350:     msg[0] = ASIM_REQ_XTEDS;
351:     msg[1] = msg[2] = 0;
352:     unsigned char pipenum = ASIM_OUT_PIPE;
353:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
3, &result, NULL))

```

```

354:     return false;
355: if(debug>=FILE_ECHO)
356: {
357:     fprintf(echo,"message sent(%ld): ",result);
358:     for(int i=0;i<3;i++)
359:         fprintf(echo,"%hhx ",msg[i]);
360:     fprintf(echo," \n");
361:     fflush(echo);
362: }
363: if (result < 3) return false;
364: return true;
365: }
366:
367: bool ASIM::TimeAtTone(long sec,long usec)
368: {
369:     unsigned char msg[16];
370:     short length = 8;
371:     unsigned long result = 0;
372:     memset(msg,0,16);
373:     msg[0] = ASIM_TIME_AT_TONE;
374:     length = htons(length);    //length needs to be big-endian
375:     PUT_SHORT(&msg[1],length);
376:     PUT_LONG(&msg[3],sec);
377:     PUT_LONG(&msg[7],usec);
378:     unsigned char pipenum = ASIM_OUT_PIPE;
379:     if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_WRITE, &pipenum, sizeof(pipenum), msg,
11, &result, NULL))
380:         return false;
381: if(debug>=FILE_ECHO)
382: {
383:     fprintf(echo,"message sent(%ld): ",result);
384:     for(int i=0;i<11;i++)
385:         fprintf(echo,"%hhx ",msg[i]);
386:     fprintf(echo," \n");
387:     fflush(echo);
388: }
389: if (result < 11) return false;
390: return true;
391: }
392:
393: int ASIM::RawRead(unsigned char* data)

```

```

394: {
395:     unsigned char msg[ASIM_MAX_OUT]; //buffer for message
396:     long result = 0;                //result of a read
397:     if (handle < 0)
398:     {
399:         if(debug>=FILE_ECHO)
400:         {
401:             fprintf(echo,"ERROR: Bad handle \n");
402:         }
403:         return ASIM_ERROR;
404:     }
405: #ifdef WIN32
406: #else
407:     result = read(handle,msg,ASIM_MAX_OUT); //read first data message
408: #endif
409:
410:     if (result < 0)
411:     {
412:         if(debug>=FILE_ECHO)
413:         {
414:             fprintf(echo,"ERROR: %s \n",strerror(errno));
415:         }
416:         return -1;
417:     }
418:     memcpy(data,msg,result);
419:     return result;
420: }
421:
422: char ASIM::Read(unsigned short& length, unsigned char* data,int buflen)
423: {
424:     unsigned char msg[ASIM_MAX_OUT]; //buffer for message
425:     unsigned char msg_type;           //first character of message
426:     unsigned long result = 0;         //result of a read
427:     int bytes_so_far;                 //number of bytes read so far
428:     length = 0;                       //if an error occurs before the length field
429:                                     //can be read return 0 as the length
430:     Sleep(1000);
431:
432:     memset(msg,0,ASIM_MAX_OUT);
433:     if (handle == NULL)
434:     {

```

```

435:     if(debug>=FILE_ECHO)
436:     {
437:         fprintf(echo,"ERROR: Bad handle \n");
438:     }
439:     return ASIM_ERROR;
440: }
441: unsigned char pipenum = ASIM_IN_PIPE;
442: if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_READ, &pipenum, sizeof(pipenum), msg,
ASIM_MAX_OUT, &result, NULL))
443:     return NULL; /*If there is nothing to read, don't report an error, just return with NULL*/
444: if (result < 0)
445: {
446:     if(debug>=FILE_ECHO)
447:     {
448:         fprintf(echo,"ERROR: %s \n",strerror(errno));
449:     }
450:     if(errno==110)
451:         return ASIM_TIMEOUT;
452:     return ASIM_ERROR;
453: }
454: msg_type = msg[0];          //save message type
455:
456: if (msg_type == ASIM_VERSION)    //version needed to know endianness of length
457: {
458:     //set version
459:     version = msg[3];
460: }
461:
462: //get length based on version
463: if(version != NO_VERSION)
464: {
465:     length = msg[2] + (msg[1] << 8);    //length is big endian, convert to host byte order
466: }
467: else
468: {
469:     length = msg[1] + (msg[2] << 8);    //length is little endian, convert to host byte order
470: }
471: bytes_so_far = result - 3;          //bytes read is the total number - header
472:
473: if (result >= 3)
474:     memcpy(data,msg+3,result-3);    //store data bytes

```

```

475:
476:  if (msg_type == ASIM_STATUS)      //status always has length 1
477:  {                                  //but may occur before a version
478:      length = 1;                  //possibly giving a wrong value of 256
479:  }
480:
481:  while(length > bytes_so_far)      //continue reading data
482:  {
483:      Sleep(100);    /*Allow time for the ASIM to get data*/
484:      pipenum = ASIM_IN_PIPE;
485:      /*If the ASIM doesn't hasn't gotten around to saying anything, it will return zero as result, and
486:      should be read again.*/
487:      if (!DeviceIoControl(handle, IOCTL_ASIM_BULK_READ, &pipenum, sizeof(pipenum),
488:      msg, ASIM_MAX_OUT, &result, NULL))
489:          continue;
490:      memcpy(data+bytes_so_far,msg,result); //store data bytes
491:      bytes_so_far+=result;
492:  }
493:
494:  if(debug>=FILE_ECHO)
495:  {
496:      fprintf(echo,"message rec'd :");
497:      for(int i=0;i<length+3;i++)
498:          fprintf(echo," %hhd",msg[i]);
499:      fprintf(echo," \n");
500:  }
501:  return msg_type;
502: }
503:
504: bool ASIM::VerifyConnection(void)
505: {
506:     HANDLE verification_handle = NULL;
507:     verification_handle = CreateFile(devicename,  GENERIC_READ|GENERIC_WRITE,
508:     FILE_SHARE_READ|FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
509:     if (verification_handle == NULL)
510:         return false;
511:     CloseHandle(verification_handle);
512:     return true;
513: }
514:
515: char* ASIM::USBLocation(void)
516: {

```

```

514:   char temp_usb[80];
515:   int start_usb_location;
516:   long result = 0;
517:
518:   //initialize
519:   memset(temp_usb,0,80);
520:   memset(usb_location,0,80);
521:   //request USB path
522: #ifndef WIN32
523:   result = ioctl(handle, ASIM_PATH, temp_usb);
524: #endif
525:
526: //strip unnecessary info
527:   for(int i = 0; i < 80; ++i) // cleanup issue 34
528:   {
529:     if (temp_usb[i]==' ')
530:     {
531:       start_usb_location = i;
532:       break;
533:     }
534:   }
535:   if (start_usb_location > 0)
536:   {
537:     ++start_usb_location;
538:     //copy to usb member variable
539:     strncpy(usb_location,temp_usb+start_usb_location,79);
540:   }
541:   return usb_location;
542: }
543:
544: void ASIM::SetDebug(int d)
545: {
546:   debug = d;
547: }
548:
549:

```

File: sdm/common/asim/asim_commands.h

```
1: #ifndef __ASIM_COMMANDS_H
2: #define __ASIM_COMMANDS_H
3:
4: //The following are codes for messages to an ASIM
5: #define ASIM_INITIALIZE      'I'
6: #define ASIM_RESET          'R'
7: #define ASIM_SELF_TEST      'T'
8: #define ASIM_REQ_DATA       'M'
9: #define ASIM_CANCEL        'C'
10: #define ASIM_REQ_STREAM     'N'
11: #define ASIM_POWER_ON       'P'
12: #define ASIM_POWER_DOWN    'F'
13: #define ASIM_REQ_VERSION 'U'
14: #define ASIM_COMMAND        'V'
15: #define ASIM_REQ_XTEDS      'X'
16: #define ASIM_TIME_AT_TONE   'O'
17:
18: //The following are codes for messages from an ASIM
19: #define ASIM_STATUS         'S'
20: #define ASIM_XTEDS          'X'
21: #define ASIM_XTEDS_ID_PAIR  'Y'
22: #define ASIM_DATA           'D'
23: #define ASIM_VERSION        'V'
24:
25: //used to return an error from the ASIM::read function
26: #define ASIM_ERROR          'e'
27: #define ASIM_TIMEOUT        't'
28:
29: #define ASIM_MAX_IN         64
30:
31: /*Win32 needs a bigger buffer to do ASIM reads*/
32: #ifdef WIN32
33: #define ASIM_MAX_OUT 4096
34: #else
35: #define ASIM_MAX_OUT      64
36: #endif
37:
38: #endif
```


File: sdm/common/semaphore/semaphore.h

```
1: #ifndef __SEM_H_
2: #define __SEM_H_
3:
4: #include <pthread.h>
5: #include <stdint.h>
6:
7: #include "../sdmLib.h"
8:
9: class SDMLIB_API Sem
10: {
11: public:
12:   Sem(int16_t i16InitialValue);
13:   Sem();
14:   Sem(const Sem &semRhs);
15:   ~Sem();
16:   Sem& operator=(const Sem &semRhs);
17:   int Signal();
18:   int Wait();
19:   int TryWait();
20:   short GetWaitCount();
21:   short Geti16Value();
22:
23: private:
24:   int16_t m_i16Value;
25:   int16_t m_i16WaitCount;
26:   pthread_mutex_t m_valueMutex;
27:   pthread_cond_t m_valueCond;
28:
29: };
30:
31: #endif
```

File: sdm/common/semaphore/semaphore.cpp

```
1: #include <stdio.h>
2: #include <errno.h>
3:
4: #include "semaphore.h"
5:
6: //define DEBUG_SEMAPHORE 1
7:
8:                                     Sem::Sem(int16_t
i16InitialValue):m_i16Value(i16InitialValue),m_i16WaitCount(0),m_valueMutex(),m_valueCond()
9: {
10:  pthread_mutex_init(&m_valueMutex, NULL);
11:  pthread_cond_init(&m_valueCond, NULL);
12:
13:
14: #ifdef DEBUG_SEMAPHORE
15:  printf("Sem::Sem::Initialized semaphore with value %d \n", m_i16Value);
16: #endif
17: }
18:
19: Sem::Sem():m_i16Value(0),m_i16WaitCount(0),m_valueMutex(),m_valueCond()
20: {
21:  Sem(0);
22: }
23:
24:                                     Sem::Sem(const          Sem
&semRhs):m_i16Value(0),m_i16WaitCount(0),m_valueMutex(semRhs.m_valueMutex),m_valueCond()
25: {
26:
27:  pthread_mutex_lock(&m_valueMutex);
28:
29:  m_i16Value = semRhs.m_i16Value;
30:  m_valueCond = semRhs.m_valueCond;
31:
32:  pthread_mutex_unlock(&m_valueMutex);
33: }
34:
35: Sem::~~Sem()
36: {
37:  pthread_mutex_destroy(&m_valueMutex);
```

```

38: pthread_cond_destroy(&m_valueCond);
39: }
40:
41: Sem& Sem::operator=(const Sem& semRhs)
42: {
43:     if (this != &semRhs)
44:     {
45:         m_i16Value = semRhs.m_i16Value;
46:         m_valueCond = semRhs.m_valueCond;
47:         m_i16WaitCount = semRhs.m_i16WaitCount;
48:     }
49:     return *this;
50: }
51:
52: int Sem::Signal()
53: {
54:     pthread_mutex_lock(&m_valueMutex);
55:     m_i16Value ++;
56: #ifdef DEBUG_SEMAPHORE
57:     printf("Sem::Signal::Value After%d WaitCount: %d \n", m_i16Value, m_i16WaitCount);
58: #endif
59:     if(m_i16Value > 0)
60:     {
61:         pthread_cond_signal(&m_valueCond);
62:         m_i16WaitCount--;
63:     }
64:     pthread_mutex_unlock(&m_valueMutex);
65:     return 0;
66: }
67:
68: int Sem::Wait()
69: {
70:     pthread_mutex_lock(&m_valueMutex);
71:     m_i16WaitCount++;
72: #ifdef DEBUG_SEMAPHORE
73:     printf("Sem::Wait::Value Before %d WaitCount: %d \n", m_i16Value, m_i16WaitCount);
74: #endif
75:     while(m_i16Value <= 0)
76:     {
77:         pthread_cond_wait(&m_valueCond, &m_valueMutex);
78:     }

```

```

79: m_i16Value--;
80: pthread_mutex_unlock(&m_valueMutex);
81: return 0;
82: }
83:
84: int Sem::TryWait()
85: {
86:     int iRetVal;
87:
88:     iRetVal = -1;
89:     pthread_mutex_lock(&m_valueMutex);
90: #ifdef DEBUG_SEMAPHORE
91:     printf("Sem::TryWait::Value Before %d \n", m_i16Value);
92: #endif
93:     if(m_i16Value > 0)
94:     {
95:         m_i16Value--;
96:         iRetVal = 0;
97:     }
98:     pthread_mutex_unlock(&m_valueMutex);
99:     return iRetVal;
100: }
101:
102: short Sem::GetWaitCount()
103: {
104:     pthread_mutex_lock(&m_valueMutex);
105:     short iRetVal = m_i16WaitCount;
106:     pthread_mutex_unlock(&m_valueMutex);
107:     return iRetVal;
108: }

```

File: sdm/common/semaphore/Makefile

```
1: #semaphore makefile
2:
3: include ../../Makefile.common
4: include ../../$(MAKEFILE_DEFS)
5:
6: .PHONY: all clean distclean
7:
8: all:  semaphore.o
9:
10: semaphore.o:  semaphore.cpp semaphore.h
11: $(CXX) $(CXXFLAGS) -fPIC -c $<
12:
13: clean:
14: rm -f *.o *~
15:
16: distclean:  clean
```

Listing from directory: sdm/dm

File: sdm/dm/SubscriptionList.cpp

```
1: #include "SubscriptionList.h"
2: #include "Subscription.h"
3: #include "../common/Debug.h"
4: #include <stdlib.h>
5: #include <string.h>
6: extern "C"
7: {
8: #include "../common/MemoryUtils.h"
9: }
10:
11: /*
12: copyList copies the subscription list pointed to by "list".
13: INPUTS:
14:     list - The list to copy
15:     tail - (output parameter) A pointer to the tail which is assigned to be the tail of the copied list
16: RETURNS:
17:     SubscriptionListNode* - A pointer to the newly copied list
18: */
19: struct SubscriptionListNode* copyList(struct SubscriptionListNode* list, struct
SubscriptionListNode** tail)
20: {
21: struct SubscriptionListNode* p;
22: struct SubscriptionListNode* head;
23: head = (struct SubscriptionListNode*)SDM_malloc(sizeof(struct SubscriptionListNode));
24: p = head;
25: for(struct SubscriptionListNode* cur=list;cur!=NULL;cur=cur->next)
26: {
27:     p->data = cur->data;
28:     if(cur->next!=NULL)
29:     {
30:         p->next = (struct SubscriptionListNode*)SDM_malloc(sizeof(struct SubscriptionListNode));
31:     }
32:     else
33:     {
34:         p->next = NULL;
35:         *tail = p;
36:     }
```

```

37:     p = p->next;
38: }
39: return head;
40: }
41:
42: /*
43: deleteList traverses through the subscription list and frees all of the items associated with each entry.
This function
44: is called from the destructor and from the assignment operator.
45: INPUTS:
46:     p - The head of the list to delete
47: RETURNS:
48:     void
49: */
50: void deleteList(struct SubscriptionListNode* p)
51: {
52: if(p==NULL) return;
53: deleteList(p->next);
54: if(p->data!=NULL)
55:     delete(p->data);
56: free(p);
57: }
58:
59: /*
60: Default Constructor, initializes the head and tail pointers to null.
61: */
62: SubscriptionList::SubscriptionList():head(NULL),tail(NULL)
63: {}
64:
65: /*
66: Copy Constructor, copies the subscription list of the copying object.
67: */
68: SubscriptionList::SubscriptionList(const SubscriptionList& b):head(NULL),tail(NULL)
69: {
70: head = copyList(b.head,&tail);
71: }
72:
73: /*
74: ~SubscriptionList (destructor) removes all subscriptions (heap allocated) in the list.
75: */
76: SubscriptionList::~~SubscriptionList()

```

```

77: {
78: deleteList(head);
79: }
80:
81: /*
82: addSubscription adds a new subscription node to the linked list of subscriptions adding the parameter
data as the Subscription
83: node.
84: INPUTS:
85:     data - The Subscription object added to the list, the pointer is added directly and not copied.
86: RETURNS:
87:     void
88: */
89: void SubscriptionList::addSubscription(Subscription* data)
90: {
91: struct SubscriptionListNode* p = (struct SubscriptionListNode*)SDM_malloc(sizeof(struct
SubscriptionListNode));
92: p->data = data;
93: p->next = NULL;
94: if(tail == NULL)
95: {
96:     head = p;
97:     tail = p;
98: }
99: else
100: {
101:     tail->next = p;
102:     tail = p;
103: }
104: }
105:
106: /*
107: addSubscription adds a new reply subscription to the list of subscriptions. This overload adds a
subscription to a DataManager
108: notification (in its xTEDS). The linked list is first checked to make sure a duplicate subscription
will not be added by
109: checking against the input parameters. If the subscription is not in the list, a new item is (malloc)
added to the linked
110: list. Debug information for this function can be output on level 4.
111: INPUTS:
112:     ip - The IP address of the subscribing process
113:     port - The port of the subscribing process

```



```

114:      mID - The message and interface identifier for the message being subscribed to
115:      debug - The debug level of the DataManager
116:  RETURNS:
117:      SubscriptionListNode* - The position in the linked list of the added SubscriptionListNode
118: */
119:      SubscriptionListNode*      SubscriptionList::addSubscription(const      SDMComponent_ID&
SubscriberId, int mID, int debug)
120: {
121:      SubscriptionListNode* result = NULL;
122:      SubscriptionListNode* node;
123:      int count = 0;
124:
125:      node = head;
126:      while(node!=NULL)    //Check to see if subscription is a duplicate
127:      {
128:          if(node->data->getInuse() == true) //Check to see if in use
129:          {
130:              if (node->data->getDestination() == SubscriberId)
131:              {
132:                  if(node->data->getmID() == mID) //Check to see if message id matches
133:                  {
134:                      if(node->data->getItems() == false)
135:                      {
136:                          debug_f(4, "Multiple subscription trying to be entered \n");
137:                          return result;    //No need to enter subscription because it is a duplicate
138:                      }
139:                  }
140:              }
141:          }
142:          node = node->next;
143:      }
144:      node = head;
145:      while(node!=NULL)
146:      {
147:          if(node->data->getInuse()==false)
148:              break;
149:          node = node->next;
150:      }
151:      if(node==NULL)
152:      {
153:          addSubscription(new Subscription());

```

```

154:     node = tail;
155: }
156: node->data->setDestination(SubscriberId);
157: node->data->setmID(mID); //Copy msg id into subscription
158: node->data->setInuse(true); //Set in use to 1
159: node->data->setItems(false); //Set Items to false to indicate there are no Items from ReqReg
because this is a DM subscription
160: debug_f(4, "Subscription is on ip: 0x%lx port: %d to message ID: 0x%x
\n",SubscriberId.getAddress(),SubscriberId.getPort(),mID);
161: result = node;
162: node = head;
163: if (debug >= 4) //only print if subscription and debug level great enough
164: {
165:     while(node!=NULL) //Print current subscriptions
166:     {
167:         if(node->data->getInuse() == true)
168:         {
169:             char SubscriberText[64];
170:             node->data->getDestination().IDToString(SubscriberText, sizeof(SubscriberText));
171:             printf("Subscription %d: %s messageID: 0x%x \n",count+1,SubscriberText,node-
>data->getmID());
172:             count++;
173:         }
174:         node = node->next;
175:     }
176: }
177: return result;
178: }
179:
180: /*
181:  addOrRemoveSubscription adds or removes a SDMReqReg or SDMSearch subscription to the
subscription list depending on the
182:  value of "reply". If the subscription is to be added, the appropriate Search or ReqReg
subscription is added according
183:  to all of the reply information. If the subscription is to be removed, the appropriate Search or
ReqReg subscription is
184:  removed based on all of the source application information.
185:  INPUTS:
186:  reply - The subscription operation, can be one of
SDM_REQREG_CURRENT_AND_FUTURE,
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS,
187:  SDM_REQREG_CANCEL, SDM_SEARCH_CURRENT_AND_FUTURE,
SDM_SEARCH_CANCEL

```

188: ip - The IP address of the subscriber

189: port - The port of the subscriber

190: source - The component identifier of the interested device or application to which the subscription is intended

191: device - The device name of the device or application to which the subscription is intended (ReqReg only)

192: interface - The interface of the device or application to which the subscription is intended (ReqReg only)

193: itemname - The itemname (ReqReg) or search query (Search) the subscriber is interested in

194: quallist - The qualifier list of the device or application to which the subscription is intended (ReqReg only)

195: ID - The identifier number for this subscription stream

196: mID - The type of subscription, can be one of SEARCH_REPLY, REQ_REG_FUTURE, or ReqRegCancellation

197: debug - The debug level of the DataManager

198: RETURNS:

199: bool - true on successful addition/removal, false otherwise

200:

201: */

202: bool SubscriptionList::addOrRemoveSubscription(int reply, const SDMComponent_ID& SubscriberId, const SDMComponent_ID& source, const char* device, const char* interface, const char* itemname, const char* quallist, int ID, int mID, int debug)

203: {

204: bool result = false;

205: static int error = 0;

206: bool cancelsingle = false;

207:

208: switch(reply) //Check the reply value

209: {

210: case SUB_CURRENT_AND_FUTURE:

211: case SUB_CURRENT_FUTURE_AND_CANCELLATIONS:

212: if(mID == SEARCH_REPLY)

213: result = addSearchSubscription(SubscriberId, source, itemname, ID, error, debug);

214: else if(mID == VAR_REQ_REPLY)

215: result = addVarReqSubscription(SubscriberId, source, interface, itemname, ID, error, debug);

216: else

217: result = addReqRegSubscription(reply, SubscriberId, source, device, interface, itemname, quallist, ID, error, debug);

218: break;

219: case SUB_CANCEL:

220: if(source.isEmpty() == true)

221: {

```

222:         if(itemname == NULL || strlen(itemname) == 0) //Check to see if the item_name
matches
223:         {
224:             if(quallist == NULL || strlen(quallist) == 0) //Check to see if the qual_list
matches
225:             {
226:                 if(device == NULL || strlen(device)==0)//Check to see if device name
matches
227:                 {
228:                     if(interface == NULL || strlen(interface)==0)//Check to see if interface
name matches
229:                     {
230:                         if(mID == SEARCH_REPLY)
231:                             result = removeSearchSubscription(SubscriberId, debug);
232:                         else if(mID == VAR_REQ_REPLY)
233:                             result = removeVarReqSubscription(SubscriberId, debug);
234:                         else
235:                             result = removeReqRegSubscription(reply, SubscriberId, debug);
236:                     }
237:                     else
238:                         cancelsingle = true;
239:                 }
240:                 else
241:                     cancelsingle = true;
242:             }
243:             else
244:                 cancelsingle = true;
245:         }
246:         else
247:             cancelsingle = true;
248:     }
249:     else
250:         cancelsingle = true;
251:     if(cancelsingle == true)
252:     {
253:         if(mID == SEARCH_REPLY)
254:             result = removeSearchSubscription(SubscriberId, source, itemname, debug);
255:         else if(mID == VAR_REQ_REPLY)
256:             result = removeVarReqSubscription(SubscriberId, source, interface, itemname,
debug);
257:         else

```

```

258:         result = removeReqRegSubscription(reply, SubscriberId, source, device,
interface, itemname, quallist, debug);
259:     }
260:     break;
261:     default:
262:         error++;
263:         printf("Unable to handle subscriptions request because the reply type of %d was invalid
Error Count: %d \n",reply,error);
264:         result = true;
265:         break;
266:     }
267:     return result;
268: }
269:
270: /*
271:  addReqRegSubscription adds a ReqReg subscription to the subscription list. This function first
checks for a duplicate subscription, if
272:  this is the case the subscription won't be added (returns true). If the subscription has not been
previously added, it is added to
273:  the subscription list (returns true). If the "reply" is set to either
SDM_REQREG_CURRENT_AND_FUTURE or SDM_SEARCH_CURRENT_AND_FUTURE
274:  one subscription is entered. If the "reply" type is set to
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS, two subscriptions are entered
275:  one corresponding to future added subscriptions, and one corresponding to future cancellations.
276:  INPUTS:
277:  reply - The subscription operation, can be one of
SDM_REQREG_CURRENT_AND_FUTURE,
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS,
278:  SDM_REQREG_CANCEL, SDM_SEARCH_CURRENT_AND_FUTURE,
SDM_SEARCH_CANCEL
279:  ip - The IP address of the subscriber
280:  port - The port of the subscriber
281:  source - The component identifier of the interested device or application to which the
subscription is intended
282:  device - The device name of the device or application to which the subscription is intended
(ReqReg only)
283:  interface - The interface of the device or application to which the subscription is intended
(ReqReg only)
284:  itemname - The itemname (ReqReg) or search query (Search) the subscriber is interested in
285:  quallist - The qualifier list of the device or application to which the subscription is intended
(ReqReg only)
286:  ID - The identifier number for this subscription stream
287:  error - (output reference parameter) incremented if a duplicate subscription exists
288:  debug - The debug level of the DataManager

```

```

289: RETURNS:
290:     bool - Always true
291:     error - (reference parameter) incremented if a duplicate subscription exists
292: */
293: bool SubscriptionList::addReqRegSubscription(int reply, const SDMComponent_ID& SubscriberId,
const SDMComponent_ID& source, const char* device, const char* interface, const char* itemname,
const char* quallist, int ID, int& error, int debug)
294: {
295:     int cont = 0, val1 = 0, val2 = 0, tempreply = 0,j;
296:     SubscriptionListNode* node;
297:     bool nosub = false;
298:
299:     if(reply == SUB_CURRENT_AND_FUTURE)
300:     {
301:         cont = 1;    //Set a variable for the number of subscriptions to be entered
302:         val1 = REQ_REG_FUTURE;
303:     }
304:     else
305:     {
306:         cont = 2;    //Set a variable for the number of subscriptions to be entered
307:         val1 = REQ_REG_FUTURE;
308:         val2 = REQ_REG_CANCELLATION;
309:     }
310:     for(j = 0; j < cont; j++)
311:     {
312:         node = head;
313:         nosub = false;
314:         while(node!=NULL)    //Check to see if subscription is a duplicate
315:         {
316:             if(node->data->getInuse() == true) //Check to see if in use
317:             {
318:                 if (node->data->getDestination() == SubscriberId)
319:                 {
320:                     if(node->data->getmID() == val1 || node->data->getmID() == val2) //Check to
see if message id matches
321:                     {
322:                         if(node->data->getSource()==source)
323:                         {
324:                             if(node->data->getItems() == true) //Check to see if message the are
items in use
325:                             {

```

```

326:                                if(strcmp(node->data->getItemName(),itemname) == 0) //Check to
see if the item_name matches
327:                                {
328:                                    if(strcmp(node->data->getQuallist(),quallist) == 0) //Check to
see if the qual_list matches
329:                                    {
330:                                        if(strcmp(node->data->getDevice(),device)==0) //Check to
see if device name matches
331:                                        {
332:                                            if(strcmp(node->data->getInterface(),interface)==0)
//Check to see if interface name matches
333:                                            {
334:                                                if(j == 0 && node->data->getmID() == val1)
335:                                                {
336:                                                    nosub = true;
337:                                                    if(reply == SUB_CURRENT_AND_FUTURE)
338:                                                    {
339:                                                        error++;
340:                                                        debug_f(2, "Multiple subscription trying to
be entered Error Count: %d \n",error);
341:                                                        return true; //No need to enter subscription
because it is a duplicate
342:                                                    }
343:                                                }
344:                                                else if(j == 1 && node->data->getmID() == val2)
345:                                                {
346:                                                    error++;
347:                                                    debug_f(2, "Multiple subscription trying to be
entered Error Count: %d \n",error);
348:                                                    return true; //No need to enter subscription
because it is a duplicate
349:                                                }
350:                                            }
351:                                        }
352:                                    }
353:                                }
354:                            }
355:                        }
356:                    }
357:                }
358:            }
359:            node = node->next;
360:        }

```

```

361:     if(nosub == false)
362:     {
363:         node = head;
364:         while(node!=NULL)    //Find an array location in the subscription list that is not in use
365:         {
366:             if(node->data->getInuse() == false)
367:                 break;
368:             node = node->next;
369:         }
370:         if(node==NULL)
371:         {
372:             addSubscription(new Subscription());
373:             node = tail;
374:         }
375:         //Enter the subscription
376:         node->data->setItemName(itemname);
377:         node->data->setQuallist(quallist);
378:         node->data->setDestination(SubscriberId);
379:         node->data->setSource(source);
380:         node->data->setDevice(device);
381:         node->data->setInterface(interface);
382:         //Set the subscription message id
383:         if(reply == SUB_CURRENT_AND_FUTURE)
384:             tempreply = REQ_REG_FUTURE;
385:         if(reply == SUB_CURRENT_FUTURE_AND_CANCELLATIONS && j == 0)
386:             tempreply = REQ_REG_FUTURE;
387:         if(reply == SUB_CURRENT_FUTURE_AND_CANCELLATIONS && j == 1)
388:             tempreply = REQ_REG_CANCELLATION;
389:         node->data->setmID(tempreply); //Copy the message id into subscription
390:         node->data->setInuse(true); //Set subscription in use
391:         node->data->setItems(true); //Set subscription items
392:         node->data->setID(ID);
393:     }
394: }
395: return true;
396: }
397:
398: /*
399:     removeReqRegSubscription removes all subscriptions corresponding to the input "port" and "ip".
    This function always returns

```



```

400:    false to notify the ReqReg function in the DM that it can finish whether or not a subscription was
cancelled.
401:    INPUTS:
402:        reply - The subscription operation, this could be SDM_REQREG_CANCEL or
SDM_SEARCH_CANCEL
403:        ip - The IP address of the subscriber to cancel
404:        port - The port of the subscriber to cancel
405:        debug - The debug level of the DataManager
406:    RETURNS:
407:        bool - Always false
408: */
409: bool SubscriptionList::removeReqRegSubscription(int reply, const SDMComponent_ID&
SubscriberId, int debug)
410: {
411:     int cont = 0, tempreply = 0,j;
412:     SubscriptionListNode* node;
413:
414:     if(reply == SUB_CANCEL)
415:         cont = 2;    //Set the number of subscriptions to cancel
416:     for(j = 0; j < cont; j++)
417:     {
418:         node = head;
419:         if(reply == SUB_CANCEL && j == 0)
420:             tempreply = REQ_REG_FUTURE; //Set the message id
421:         if(reply == SUB_CANCEL && j == 1)
422:             tempreply = REQ_REG_CANCELLATION;    //Set the message id
423:         while(node!=NULL)
424:         {
425:             if(node->data->getInuse() == true) //Check to see if subscription in use
426:             {
427:                 SDMComponent_ID tempId = node->data->getDestination();
428:                 if (tempId.getAddress() == SubscriberId.getAddress() && tempId.getPort() ==
SubscriberId.getPort())
429:                 {
430:                     if(node->data->getmID() == tempreply) //Check to see if message ID match
431:                     {
432:                         if(node->data->getItems() == true) //Check to see it item is set
433:                         {
434:                             node->data->setPort(0); //Set port to 0
435:                             node->data->setmID(0); //Set message id to 0
436:                             node->data->setInuse(false);    //Set in use to 0
437:                             node->data->setItems(false);    //Set items to 0

```

```

438:                node->data->setID(0);
439:                debug_f(4, "Canceled subscription \n");
440:            }
441:        else
442:            debug_f(4, "Unable to cancel subscription because there is no items \n");
443:        }
444:    else
445:        debug_f(4, "Unable to cancel subscription because message ID does not
match current subscriptions \n");
446:    }
447:    else
448:        debug_f(4, "Unable to cancel subscription because address does not match
current subscription \n");
449:    }
450:    node = node->next;
451: }
452: }
453: return false;
454: }
455:
456: /*
457:  removeReqRegSubscription removes a SDMReqReg subscription according to the paramemter
information of the subscription. This function
458:  removes at max a matching pair of subscriptions. This function always returns false to notify the
ReqReg function in the DM that it
459:  can finish whether or not a subscription was cancelled.
460:  INPUTS:
461:  reply - The subscription operation, can be one of
SDM_REQREG_CURRENT_AND_FUTURE,
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS,
462:  SDM_REQREG_CANCEL,          SDM_SEARCH_CURRENT_AND_FUTURE,
SDM_SEARCH_CANCEL
463:  ip - The IP address of the subscriber
464:  port - The port of the subscriber
465:  source - The component identifier of the interested device or application to which the
subscription is intended
466:  device - The device name of the device or application to which the subscription is intended
(ReqReg only)
467:  interface - The interface of the device or application to which the subscription is intended
(ReqReg only)
468:  itemname - The itemname (ReqReg) or search query (Search) the subscriber is interested in
469:  quallist - The qualifier list of the device or application to which the subscription is intended
(ReqReg only)

```

```

470:     debug - The debug level of the DataManager
471: RETURNS:
472:     bool - Always false
473:
474: */
475: bool SubscriptionList::removeReqRegSubscription(int reply, const SDMComponent_ID&
SubscriberId, const SDMComponent_ID& source, const char* device, const char* interface, const char*
itemname, const char* quallist, int debug)
476: {
477:     int cont = 0, tempreply = 0,j;
478:     SubscriptionListNode* node;
479:
480:     if (device==NULL || interface==NULL || itemname==NULL || quallist==NULL)
481:         return false;
482:     if(reply == SUB_CANCEL)
483:         cont = 2;    //Set the number of subscriptions to cancel
484:     for(j = 0; j < cont; j++)
485:     {
486:         node = head;
487:         if(reply == SUB_CANCEL && j == 0)
488:             tempreply = REQ_REG_FUTURE; //Set the message id
489:         if(reply == SUB_CANCEL && j == 1)
490:             tempreply = REQ_REG_CANCELLATION;    //Set the message id
491:         while(node!=NULL)
492:         {
493:             if(node->data->getInuse() == true) //Check to see if subscription in use
494:             {
495:                 if (node->data->getDestination() == SubscriberId)
496:                 {
497:                     if(node->data->getmID() == tempreply) //Check to see if message ID match
498:                     {
499:                         if(node->data->getItems() == true) //Check to see it item is set
500:                         {
501:                             if(node->data->getSource()==source)
502:                             {
503:                                 if(strcmp(node->data->getItemName(),itemname) == 0) //Check to
see if the item_name matches
504:                                 {
505:                                     if(strcmp(node->data->getQuallist(),quallist) == 0) //Check to
see if the qual_list matches
506:                                     {

```

```

507:                                     if(strcmp(node->data->getDevice(),device)==0) //Check to
see if device name matches
508:                                     {
509:                                     if(strcmp(node->data->getInterface(),interface)==0)
//Check to see if interface name matches
510:                                     {
511:                                     node->data->setPort(0); //Set port to 0
512:                                     node->data->setmID(0); //Set message id to 0
513:                                     node->data->setInuse(false); //Set in use to 0
514:                                     node->data->setItems(false); //Set items to 0
515:                                     node->data->setID(0);
516:                                     debug_f(4, "Canceled subscription \n");
517:                                     if(reply == SUB_CANCEL && j == 1)
518:                                     return false; //If subscriptions canceled this
function is done
519:                                     if(reply < SUB_CANCEL)
520:                                     return false; //If subscription is canceled this
function is done
521:                                     }
522:                                     else
523:                                     debug_f(4, "Unable to cancel subscription because
there is no interface \n");
524:                                     }
525:                                     else
526:                                     debug_f(4, "Unable to cancel subscription because there
is no device \n");
527:                                     }
528:                                     else
529:                                     debug_f(4, "Unable to cancel subscription because there is
no quallist \n");
530:                                     }
531:                                     else
532:                                     debug_f(4, "Unable to cancel subscription because there is no
item_name \n");
533:                                     }
534:                                     else
535:                                     debug_f(4, "Unable to sources do not match \n");
536:                                     }
537:                                     else
538:                                     debug_f(4, "Unable to cancel subscription because there is no items \n");
539:                                     }
540:                                     else

```

```

541:             debug_f(4, "Unable to cancel subscription because message ID does not
match current subscriptions \n");
542:         }
543:         else
544:             debug_f(4, "Unable to cancel subscription because address does not match
current subscription \n");
545:     }
546:     node = node->next;
547: }
548: }
549: return false;
550: }
551:
552: /*
553:  addSearchSubscription adds a subscription for SDMSearch requests according to the input
information. This function first checks
554:  to be sure that the current request is not a duplicate subscription.
555:  INPUTS:
556:      ip - The IP address of the subscriber
557:      port - The port of the subscriber
558:      source - The component identifier of the interested device or application to which the
subscription is intended
559:      itemname - The itemname (ReqReg) or search query (Search) the subscriber is interested in
560:      ID - The identifier number for this subscription stream
561:      error - (output reference parameter) incremented if a duplicate subscription exists
562:      debug - The debug level of the DataManager
563:  RETURNS:
564:      bool - Always true
565: */
566: bool SubscriptionList::addSearchSubscription(const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* itemname, int ID, int& error, int debug)
567: {
568:     SubscriptionListNode* node;
569:
570:     node = head;
571:     while(node!=NULL)    //Check to see if subscription is a duplicate
572:     {
573:         if(node->data->getInuse() == true) //Check to see if in use
574:         {
575:             if(node->data->getDestination() == SubscriberId)
576:             {
577:                 if(node->data->getSource()==source)

```

```

578:         {
579:             if(node->data->getItems() == true) //Check to see if message the are items in
use
580:                 {
581:                     if(strcmp(node->data->getItemName(),itemname) == 0) //Check to see if the
item_name matches
582:                         {
583:                             if(node->data->getmID() == SEARCH_REPLY)
584:                                 {
585:                                     error++;
586:                                     debug_f(2, "Multiple subscription trying to be entered Error Count:
%d \n",error);
587:                                     return true; //No need to enter subscription because it is a duplicate
588:                                 }
589:                         }
590:                 }
591:             }
592:         }
593:     }
594:     node = node->next;
595: }
596: node = head;
597: while(node!=NULL) //Find an array location in the subscription list that is not in use
598: {
599:     if(node->data->getInuse() == false)
600:         break;
601:     node = node->next;
602: }
603: if(node==NULL)
604: {
605:     addSubscription(new Subscription());
606:     node = tail;
607: }
608: //Enter the subscription
609: node->data->setItemName(itemname);
610: node->data->setDestination(SubscriberId);
611: node->data->setSource(source);
612: node->data->setmID(SEARCH_REPLY); //Copy the message id into subscription
613: node->data->setInuse(true); //Set subscription in use
614: node->data->setItems(true); //Set subscription items
615: node->data->setID(ID);
616: return true;

```

```

617: }
618:
619: /*
620:  removeSearchSubscription removes a subscription from an SDMSearch query.
621:  INPUTS:
622:      ip - The IP address of the subscriber
623:      port - The port of the subscriber
624:      debug - The debug level of the DataManager
625:  RETURNS:
626:      bool - Always false
627: */
628: bool SubscriptionList::removeSearchSubscription(const SDMComponent_ID& SubscriberId, int
debug)
629: {
630:     SubscriptionListNode* node;
631:
632:     node = head;
633:     while(node!=NULL)
634:     {
635:         if(node->data->getInuse() == true) //Check to see if subscription in use
636:         {
637:             if (node->data->getDestination() == SubscriberId)
638:             {
639:                 if(node->data->getmID() == SEARCH_REPLY) //Check to see if message ID
match
640:                 {
641:                     if(node->data->getItems() == true) //Check to see it item is set
642:                     {
643:                         node->data->setPort(0); //Set port to 0
644:                         node->data->setmID(0); //Set message id to 0
645:                         node->data->setInuse(false); //Set in use to 0
646:                         node->data->setItems(false); //Set items to 0
647:                         node->data->setID(0);
648:                         debug_f(4, "Canceled subscription \n");
649:                     }
650:                     else
651:                         debug_f(4, "Unable to cancel subscription because there is no items \n");
652:                 }
653:                 else
654:                     debug_f(4, "Unable to cancel subscription because message ID does not match
current subscriptions \n");
655:             }

```

```

656:         else
657:             debug_f(4, "Unable to cancel subscription because address does not match current
subscription \n");
658:         }
659:         node = node->next;
660:     }
661:     return false;
662: }
663:
664: /*
665:  removeSearchSubscription removes a subscription from an SDMSearch query.
666:  INPUTS:
667:      ip - The IP address of the subscriber
668:      port - The port of the subscriber
669:      source - The component identifier of the interested sensor or application
670:      itemname - The search query issued
671:      debug - The debug level of the DataManager
672:  RETURNS:
673:      bool - Always false
674:  */
675: bool SubscriptionList::removeSearchSubscription(const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* itemname, int debug)
676: {
677:     SubscriptionListNode* node;
678:
679:     if (itemname==NULL)
680:         return false;
681:
682:     node = head;
683:     while(node!=NULL)
684:     {
685:         if(node->data->getInuse() == true) //Check to see if subscription in use
686:         {
687:             if (node->data->getDestination() == SubscriberId)
688:             {
689:                 if(node->data->getmID() == SEARCH_REPLY) //Check to see if message ID
match
690:                 {
691:                     if(node->data->getSource()==source)
692:                     {
693:                         if(strcmp(node->data->getItemName(),itemname) == 0) //Check to see if the
item_name matches

```



```

694:         {
695:             node->data->setPort(0); //Set port to 0
696:             node->data->setmID(0); //Set message id to 0
697:             node->data->setInuse(false);    //Set in use to 0
698:             node->data->setItems(false);    //Set items to 0
699:             node->data->setID(0);
700:             debug_f(4, "Canceled subscription \n");
701:             return false;
702:         }
703:         else
704:             debug_f(4, "Unable to cancel subscription because there is no item_name
\n");
705:     }
706:     else
707:         debug_f(4, "Unable to sources do not match \n");
708:     }
709:     else
710:         debug_f(4, "Unable to cancel subscription because message ID does not match
current subscriptions \n");
711:     }
712:     else
713:         debug_f(4, "Unable to cancel subscription because address does not match current
subscription \n");
714:     }
715:     node = node->next;
716: }
717: return false;
718: }
719:
720: /*
721:  addVarReqSubscription adds a subscription for SDMSearch requests according to the input
information. This function first checks
722:  to be sure that the current request is not a duplicate subscription.
723:  INPUTS:
724:      ip - The IP address of the subscriber
725:      port - The port of the subscriber
726:      source - The component identifier of the interested device or application to which the
subscription is intended
727:      interface - The string version of the message id interface pair
728:      itemname - The variable name
729:      ID - The identifier number for this subscription stream
730:      error - (output reference parameter) incremented if a duplicate subscription exists

```

```

731:      debug - The debug level of the DataManager
732:  RETURNS:
733:      bool - Always true
734:  */
735:  bool SubscriptionList::addVarReqSubscription(const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* interface, const char* itemname, int ID, int& error, int debug)
736:  {
737:      SubscriptionListNode* node;
738:
739:      node = head;
740:      while(node!=NULL)    //Check to see if subscription is a duplicate
741:      {
742:          if(node->data->getInuse() == true) //Check to see if in use
743:          {
744:              if (node->data->getDestination() == SubscriberId)
745:              {
746:                  if(node->data->getSource()==source)
747:                  {
748:                      if(node->data->getItems() == true) //Check to see if message the are items in
use
749:                      {
750:                          if(strcmp(node->data->getItemName(),itemname) == 0) //Check to see if the
item_name matches
751:                          {
752:                              if(strcmp(node->data->getInterface(),interface)==0) //Check to see if
interface name matches
753:                              {
754:                                  if(node->data->getmID() == VAR_REQ_REPLY)
755:                                  {
756:                                      error++;
757:                                      debug_f(2, "Multiple subscription trying to be entered Error
Count: %d \n",error);
758:                                      return true; //No need to enter subscription because it is a
duplicate
759:                                  }
760:                              }
761:                          }
762:                      }
763:                  }
764:              }
765:          }
766:          node = node->next;

```

```

767:     }
768:     node = head;
769:     while(node!=NULL)    //Find an array location in the subscription list that is not in use
770:     {
771:         if(node->data->getInuse() == false)
772:             break;
773:         node = node->next;
774:     }
775:     if(node==NULL)
776:     {
777:         addSubscription(new Subscription());
778:         node = tail;
779:     }
780:     //Enter the subscription
781:     node->data->setItemName(itemname);
782:     node->data->setDestination(SubscriberId);
783:     node->data->setSource(source);
784:     node->data->setmID(VAR_REQ_REPLY); //Copy the message id into subscription
785:     node->data->setInuse(true); //Set subscription in use
786:     node->data->setItems(true); //Set subscription items
787:     node->data->setID(ID);
788:     node->data->setInterface(interface);
789:     return true;
790: }
791:
792: /*
793:  removeVarReqSubscription removes a subscription from an SDMSearch query.
794:  INPUTS:
795:      ip - The IP address of the subscriber
796:      port - The port of the subscriber
797:      debug - The debug level of the DataManager
798:  RETURNS:
799:      bool - Always false
800:  */
801: bool SubscriptionList::removeVarReqSubscription(const SDMComponent_ID& SubscriberId, int
debug)
802: {
803:     SubscriptionListNode* node;
804:
805:     node = head;
806:     while(node!=NULL)

```

```

807:  {
808:      if(node->data->getInuse() == true) //Check to see if subscription in use
809:      {
810:          if (node->data->getDestination() == SubscriberId)
811:          {
812:              if(node->data->getmID() == VAR_REQ_REPLY) //Check to see if message ID
match
813:              {
814:                  if(node->data->getItems() == true) //Check to see if item is set
815:                  {
816:                      node->data->setPort(0); //Set port to 0
817:                      node->data->setmID(0); //Set message id to 0
818:                      node->data->setInuse(false); //Set in use to 0
819:                      node->data->setItems(false); //Set items to 0
820:                      node->data->setID(0);
821:                      debug_f(4, "Canceled subscription \n");
822:                  }
823:                  else
824:                      debug_f(4, "Unable to cancel subscription because there is no items \n");
825:              }
826:              else
827:                  debug_f(4, "Unable to cancel subscription because message ID does not match
current subscriptions \n");
828:          }
829:          else
830:              debug_f(4, "Unable to cancel subscription because address does not match current
subscription \n");
831:          }
832:          node = node->next;
833:      }
834:      return false;
835: }
836:
837: /*
838:  removeVarReqSubscription removes a subscription from an SDMSearch query.
839:  INPUTS:
840:      ip - The IP address of the subscriber
841:      port - The port of the subscriber
842:      source - The component identifier of the interested sensor or application
843:      interface - The string version of the message id interface pair
844:      itemname - The variable name
845:      debug - The debug level of the DataManager

```

```

846: RETURNS:
847:     bool - Always false
848: */
849: bool SubscriptionList::removeVarReqSubscription(const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* interface, const char* itemname, int debug)
850: {
851:     SubscriptionListNode* node;
852:
853:     if (itemname==NULL)
854:         return false;
855:
856:     node = head;
857:     while(node!=NULL)
858:     {
859:         if(node->data->getInuse() == true) //Check to see if subscription in use
860:         {
861:             if (node->data->getDestination() == SubscriberId)
862:             {
863:                 if(node->data->getmID() == VAR_REQ_REPLY) //Check to see if message ID
match
864:                 {
865:                     if(node->data->getSource()==source)
866:                     {
867:                         if(strcmp(node->data->getItemName(),itemname) == 0) //Check to see if the
item_name matches
868:                         {
869:                             node->data->setPort(0); //Set port to 0
870:                             node->data->setmID(0); //Set message id to 0
871:                             node->data->setInuse(false); //Set in use to 0
872:                             node->data->setItems(false); //Set items to 0
873:                             node->data->setID(0);
874:                             debug_f(4, "Canceled subscription \n");
875:                             return false;
876:                         }
877:                         else
878:                             debug_f(4, "Unable to cancel subscription because there is no item_name
\n");
879:                     }
880:                     else
881:                         debug_f(4, "Unable to sources do not match \n");
882:                 }
883:             }

```

```

884:             debug_f(4, "Unable to cancel subscription because message ID does not match
current subscriptions \n");
885:         }
886:     else
887:         debug_f(4, "Unable to cancel subscription because address does not match current
subscription \n");
888:     }
889:     node = node->next;
890: }
891: return false;
892: }
893:
894: /*
895:  operator= (Assignment operator) deletes the current instance's subscription list and copies the
subscription list of the
896:  right operand.
897:  INPUTS:
898:      b - The right operand
899:  RETURNS:
900:      SubscriptionList& - Reference to "this"
901: */
902: SubscriptionList& SubscriptionList::operator=(const SubscriptionList& b)
903: {
904:     deleteList(head);
905:     head = copyList(b.head,&tail);
906:     return *this;
907: }
908:

```

File: sdm/dm/DMUtils.h

```
1: // Extra utility functions used by the Data Manager
2:
3: #ifndef _SDM_DM_UTIL_H_
4: #define _SDM_DM_UTIL_H_
5:
6: #define SdmImaDm  'd'
7: #define SdmImaTm  't'
8:
9: #ifndef WIN32
10: # include <net/if.h>
11: # include <netdb.h>
12: #endif
13:
14: #ifdef __uLinux__
15: #warning "Including SendIMA"
16: #define SEND_IMA
17: #include <netspwnp/spwnp.h>
18: struct spwnp_ima {
19:  struct spwnp_hdr    hdr;
20:  unsigned long       ip;
21:  unsigned short      port;
22:  unsigned char       csum;
23: } __attribute__((packed));
24:
25: #endif
26:
27: unsigned long GetNodeAddress(bool spacewire=false);
28:
29: #ifdef SEND_IMA
30: int SendIMA(unsigned char which, int debug);
31: #endif
32:
33: bool IsPIDFromASIM (long processID);
34: bool PIDToFileName (long processID, char fileName[]);
35: int Store_xTEDS (long processID, char* xTEDSIn);
36: int Retrieve_xTEDS (long processID, char* xTEDSOut);
37: void CreatexTEDSDirectory();
38:
39: #endif //SDM_DM_UTILS
```

File: sdm/dm/DMxTEDS.h

```
1: #ifndef _SDM_DM_XTEDS_H_
2: #define _SDM_DM_XTEDS_H_
3:
4: /*The Data Mangers xTED*/
5: const char *dmxTED = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n \
6:     <xTEDS      xmlns=      \"http://www.interfacecontrol.com/SPA/xTEDS      \"      xmlns:xsi=
7:     \"http://www.w3.org/2001/XMLSchema-instance      \"      xsi:schemaLocation=
8:     \"http://www.interfacecontrol.com/SPA/xTEDS      ../Schema/xTEDS02.xsd      \"      name=
9:     \"Data_Manager_xTEDS \" version= \"2.0 \"> \n \
10: \t<Application name= \"DataManager \" kind= \"Software \"/> \n \
11: \t<Interface name= \"DM_Interface \" id= \"1 \"> \n \
12: \t\t<Variable name= \"Type \" kind= \"Type_of_Response \" format= \"UINT08 \"> \n \
13: \t\t\t<Drange name= \"Type_of_Response_Mode \"> \n \
14: \t\t\t\t<Option name= \"Register \" value= \"1 \"/> \n \
15: \t\t\t\t<Option name= \"Deregister \" value= \"2 \"/> \n \
16: \t\t\t\t<Option name= \"Modification \" value= \"3 \"/> \n \
17: \t\t\t</Drange> \n \
18: \t\t</Variable> \n \
19: \t\t<Variable name= \"Sensor_ID \" kind= \"ID \" format= \"UINT32 \"/> \n \
20: \t\t<Variable format= \"UINT08 \" name= \"DeviceName \" kind= \"String \" length= \"81 \"/> \n \
21: \t\t<Variable format= \"UINT08 \" name= \"SPANodePath \" kind= \"Location_of_Device \" length=
22: \t\t\"80 \"/> \n \
23: \t\t<Variable format= \"UINT32 \" name= \"pid \" kind= \"Process_ID \"/> \n \
24: \t\t<Variable format= \"UINT16 \" name= \"Port \" kind= \"Port_of_Device \"/> \n \
25: \t\t<Variable format= \"UINT32 \" name= \"Address \" kind= \"IP_long \"/> \n \
26: \t\t<Variable format= \"UINT08 \" name= \"ComponentKey \" kind= \"String \" length= \"129 \"/> \n
27: \
28: \n \
29: \t\t<Notification> \n \
30: \t\t\t\t<DataMsg name= \"Registration \" id= \"1 \" msgArrival= \"EVENT \" description=
31: \t\t\t\t\"Registration Event \"> \n \
32: \t\t\t\t\t<VariableRef name= \"Sensor_ID \"/> \n \
33: \t\t\t\t</DataMsg> \n \
34: \t\t</Notification> \n \
35: \t\t<Notification> \n \
36: \t\t\t\t<DataMsg name= \"Deregistration \" id= \"2 \" msgArrival= \"EVENT \" description=
37: \t\t\t\t\"Deregistration Event \"> \n \
38: \t\t\t\t\t<VariableRef name= \"Sensor_ID \"/> \n \
39: \t\t\t\t</DataMsg> \n \
40: \t\t</Notification> \n \
```


34: \t \t<Notification> \n \
 35: \t \t \t<DataMsg name= \"Modification \" id= \"3 \" msgArrival= \"EVENT \" description= \"xTEDS
 Update or Merging after registration event \">> \n \
 36: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
 37: \t \t \t</DataMsg> \n \
 38: \t \t</Notification> \n \
 39: \t \t<Notification> \n \
 40: \t \t \t<DataMsg name= \"RegisterChange \" id= \"4 \" msgArrival= \"EVENT \" description= \"Any
 Registration or Deregistration or Modification Change \">> \n \
 41: \t \t \t \t<VariableRef name= \"Type \"/> \n \
 42: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
 43: \t \t \t</DataMsg> \n \
 44: \t \t</Notification> \n \
 45: \n \
 46: \t \t<Request> \n \
 47: \t \t \t<CommandMsg name= \"SendSensorID \" id= \"6 \">> \n \
 48: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
 49: \t \t \t</CommandMsg> \n \
 50: \t \t \t<DataReplyMsg name= \"ConvertedDeviceName \" id= \"5 \">> \n \
 51: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
 52: \t \t \t \t<VariableRef name= \"DeviceName \"/> \n \
 53: \t \t \t</DataReplyMsg> \n \
 54: \t \t</Request> \n \
 55: \t \t<Request> \n \
 56: \t \t \t<CommandMsg name= \"SensorIDtoSPANode \" id= \"11 \">> \n \
 57: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
 58: \t \t \t</CommandMsg> \n \
 59: \t \t \t<DataReplyMsg name= \"ConvertedSPANode \" id= \"7 \">> \n \
 60: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
 61: \t \t \t \t<VariableRef name= \"SPANodePath \"/> \n \
 62: \t \t \t</DataReplyMsg> \n \
 63: \t \t</Request> \n \
 64: \t \t<Request> \n \
 65: \t \t \t<CommandMsg name= \"SendPID \" id= \"8 \">> \n \
 66: \t \t \t \t<VariableRef name= \"pid \"/> \n \
 67: \t \t \t</CommandMsg> \n \
 68: \t \t \t<DataReplyMsg name= \"ReturnSensorID \" id= \"9 \">> \n \
 69: \t \t \t \t<VariableRef name= \"pid \"/> \n \
 70: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
 71: \t \t \t</DataReplyMsg> \n \
 72: \t \t</Request> \n \

```

73: \t \t<Request> \n \
74: \t \t \t<CommandMsg name= \"SensorIDtoIP \" id= \"12 \"> \n \
75: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
76: \t \t \t</CommandMsg> \n \
77: \t \t \t<DataReplyMsg name= \"ConvertedIP \" id= \"10 \"> \n \
78: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
79: \t \t \t \t<VariableRef name= \"Address \"/> \n \
80: \t \t \t \t<VariableRef name= \"Port \"/> \n \
81: \t \t \t</DataReplyMsg> \n \
82: \t \t</Request> \n \
83: \t \t<Request> \n \
84: \t \t \t<CommandMsg name= \"ComponentIDtoComponentKey \" id= \"14 \"> \n \
85: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
86: \t \t \t \t<VariableRef name= \"Address \"/> \n \
87: \t \t \t \t<VariableRef name= \"Port \"/> \n \
88: \t \t \t</CommandMsg> \n \
89: \t \t \t<DataReplyMsg name= \"ReturnComponentKey \" id= \"15 \"> \n \
90: \t \t \t \t<VariableRef name= \"ComponentKey \"/> \n \
91: \t \t \t</DataReplyMsg> \n \
92: \t \t</Request> \n \
93: \t</Interface> \n \
94: \n \
95: \t<Interface name= \"Msg_Count \" id= \"2 \"> \n \
96: \t \t<Variable name= \"Total_Messages_Recd \" kind= \"Total \" format= \"UINT32 \"/> \n \
97: \t \t<Variable name= \"Messages_Last_Second_Recd \" kind= \"Total \" format= \"UINT32 \"/> \n \
98: \t \t<Variable name= \"Total_Messages_Sent \" kind= \"Total \" format= \"UINT32 \"/> \n \
99: \t \t<Variable name= \"Messages_Last_Second_Sent \" kind= \"Total \" format= \"UINT32 \"/> \n \
100: \t \t<Variable name= \"DroppedxTEDS \" kind= \"counter \" format= \"UINT32 \"/> \n \
101: \t \t<Variable name= \"DroppedCancelxTEDS \" kind= \"counter \" format= \"UINT32 \"/> \n \
102: \n \
103: \t \t<Notification> \n \
104: \t \t \t<DataMsg name= \"Message_Count \" id= \"13 \" msgArrival= \"PERIODIC \"> \n \
105: \t \t \t \t<VariableRef name= \"Total_Messages_Recd \"/> \n \
106: \t \t \t \t<VariableRef name= \"Messages_Last_Second_Recd \"/> \n \
107: \t \t \t \t<VariableRef name= \"Total_Messages_Sent \"/> \n \
108: \t \t \t \t<VariableRef name= \"Messages_Last_Second_Sent \"/> \n \
109: \t \t \t</DataMsg> \n \
110: \t \t</Notification> \n \
111: \t \t<Request> \n \
112: \t \t \t<CommandMsg name= \"GetErrors \" id= \"18 \" /> \n \
113: \t \t \t<DataReplyMsg name= \"ErrorReply \" id= \"19 \" > \n \

```

```

114: \t \t \t <VariableRef name= \"DroppedxTEDS \" /> \n \
115: \t \t \t <VariableRef name= \"DroppedCancelxTEDS \" /> \n \
116: \t \t </DataReplyMsg> \n \
117: \t </Request> \n \
118: </Interface> \n \
119: \t <Interface name= \"Message_Log \" id= \"3 \"> \n \
120: \t \t <Variable format= \"UINT08 \" name= \"Msg_Type \" kind= \"TBD \" /> \n \
121: \t \t <Variable format= \"UINT32 \" name= \"Address \" kind= \"IP_long \" /> \n \
122: \t \t <Variable format= \"UINT16 \" name= \"Port \" kind= \"Port_of_Device \" /> \n \
123: \t \t <Variable format= \"UINT32 \" name= \"Sensor_ID \" kind= \"ID \" /> \n \
124: \t \t <Command> \n \
125: \t \t \t <CommandMsg name= \"Enable_Logging \" id= \"16 \"> \n \
126: \t \t \t <VariableRef name= \"Msg_Type \" /> \n \
127: \t \t \t <VariableRef name= \"Address \" /> \n \
128: \t \t \t <VariableRef name= \"Port \" /> \n \
129: \t \t \t <VariableRef name= \"Sensor_ID \" /> \n \
130: \t \t </CommandMsg> \n \
131: \t </Command> \n \
132: \t \t <Command> \n \
133: \t \t \t <CommandMsg name= \"Disable_Logging \" id= \"17 \"> \n \
134: \t \t \t <VariableRef name= \"Msg_Type \" /> \n \
135: \t \t \t <VariableRef name= \"Address \" /> \n \
136: \t \t \t <VariableRef name= \"Port \" /> \n \
137: \t \t \t <VariableRef name= \"Sensor_ID \" /> \n \
138: \t \t </CommandMsg> \n \
139: \t </Command> \n \
140: </Interface> \n \
141: </xTEDS> \n";
142:
143: #endif

```

File: sdm/dm/DM.cpp

```
1: //DM.cpp
2: /*
3:  The Data Manager (DM) module of the Satellite Data Model (SDM) system is responsible
4:  for maintaining information about the data sources and service providers available
5:  within the SDM system. The DM provides a mechanism for user applications to find data
6:  and service providers registered with the system, subscribe to data, and issue commands.
7:  The DM also provides a mechanism for data and service providers to register their
8:  capabilities in the SDM system, and handle consumer application requests.
9:
10: The DM:
11:  - Interfaces via TCP/UDP and system calls.
12:  - Uses pThreads, File access,
13:  -
14: */
15: #include <stdio.h>
16: #include <stdlib.h>
17: #include <string.h>
18: #include <fcntl.h>
19: #include <unistd.h>
20: #include <arpa/inet.h>
21: #include <ctype.h>
22: #ifndef __VXWORKS__
23: #include <getopt.h>
24: #include <sys/poll.h>
25: #endif
26: #include <signal.h>
27: #include <pthread.h>
28: #include <sys/socket.h>
29: #include <sys/time.h>
30: #include <sys/ioctl.h>
31:
32: #include "../common/Time/SDMTime.h"
33: #include "../common/task/taskdefs.h"
34: #ifdef __VXWORKS__
35: #include <ioLib.h>
36: #include <pipeDrv.h>
37: #include <vxWorks.h>
38: #include <time.h>
39: #endif
```

```

40:
41: #ifdef WIN32
42: #include <Winsock2.h>
43: #else
44: #include <net/if.h>
45: #include <netdb.h>
46: #include <sys/stat.h>
47: #endif
48:
49: #include "DM.h"
50: #include "DMxTEDS.h"
51:
52:
53: #define xsize 25
54: #define DM_STACK_SIZE (256000)
55:
56: #ifdef PNP_BACKUP
57: SDMComponent_ID NetworkManager;
58: int dm_heartbeat_pipe[2];
59: #endif
60:
61: Sem writeSem(1); //Semaphore used for writing/reading from the pipe
62: Sem addLibrary(1); //Semaphore used for registering/deregistering xTEDS
63: const int MAX_OUTSTANDING_XTEDS = 5; // The number of outstanding xTEDS registrations
    allowed (as a semaphore counter)
64:
65: //xTEDS segment builder used to build segmented UDP xTEDS documents, all threads
66: //must first lock the SegmentBuilderMutex to access the class
67: xTEDSSegmentBuilder xTEDSBuilder;
68: pthread_mutex_t SegmentBuilderMutex = PTHREAD_MUTEX_INITIALIZER;
69:
70: //These socket descriptors are thread safe because any problems that might occur will only occur
    when a SIGINT has
71: // happened and the DM needs to be quitting anyway
72:      int      udpSock=IP_SOCK_INVALID,      tcpSock=IP_SOCK_INVALID,
tcpListenSock=IP_SOCK_INVALID;
73:
74: //The write operations into the pipes is protected by the writeSem semaphore variable, only the
    ChildFunctionCallProcess reads from the pipes
75: int MsgTxPushSock=IP_SOCK_INVALID;
76: #ifndef __VXWORKS__
77: int alert[2]; /*Pipes for parent/child communication*/

```

```

78: #else //VxWorks doesn't support UNIX-style pipes, a named pipe must be used
79: int alertPipe;
80: #endif
81:
82: //debug is only written upon initialization from command line parameters, all other accesses are reads
83: int debug = 0;          /*Global debug level so functions know which printf's to print*/
84:
85: //spacewire is only written upon initialization from command line parameters, all other accesses are reads
86: bool spacewire = false;
87:
88: //Each element in the list has an associated semaphore called inUse that must be held to modify or
read any of the data
89: xTEDSLibraryList xTEDSList;
90:
91: //Protected by the subscription_mutex, this mutex must be held to modify or read any data from
subscribers
92: SubscriptionList subscribers;
93: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER;
94:
95: //Protected by the log_service_mutex, this mutex must be held to modify or read any data from
log_service
96: MessageLogger log_service;
97: pthread_mutex_t log_service_mutex = PTHREAD_MUTEX_INITIALIZER;
98:
99: //Queues used during the registration process, protected by mutexes.
100: SDMRegQueue deviceQueue;
101: SDMRegQueue applicationQueue;
102: pthread_mutex_t reg_queue_mutex = PTHREAD_MUTEX_INITIALIZER;
103: bool registerNext = true;
104:
105: SDMCancelQueue cancelxTEDSQueue;
106: pthread_mutex_t cancel_queue_mutex = PTHREAD_MUTEX_INITIALIZER;
107: bool cancelNext = true;
108:
109: //Protected by the dm_address_mutex, this mutex must be held to modify or read any data from
Address_DM
110: const unsigned long L_LOCAL_HOST = inet_addr("127.0.0.1");
111: unsigned long Address_DM = L_LOCAL_HOST;
112: const unsigned long DM_SENSOR_ID = 1;
113:

```

```

114: //These message counters are protected by the perf_counter_mutex, it must be held to modify or read
the performance counters
115: unsigned int total_recvd = 0;    //message counter for total received for life of dm
116: unsigned int prevsec_recvd = 0;    //message counter for total received previous second
117: unsigned int total_sent = 0;    //message counter for total sent for life of dm
118: unsigned int prevsec_sent = 0;    //message counter for total sent previous second
119: unsigned int droppedxTEDS = 0;    //message counter for total sent previous second
120: unsigned int droppedCancelxTEDS = 0; //message counter for total sent previous second
121: pthread_mutex_t perf_counter_mutex = PTHREAD_MUTEX_INITIALIZER;
122:
123: //Sensor subscriptions
124: ProviderSubscriptionList g_SensorSubscriptions;
125: pthread_mutex_t sensor_subs_mutex = PTHREAD_MUTEX_INITIALIZER;
126:
127: #ifdef PNP_BACKUP
128: //Protected by the dm_list_mutex, this mutex must be held to modify or read any data from dmList
129: bool g_bElectedDm = true;
130: DMBackupList backupDMList;
131: pthread_mutex_t dm_list_mutex = PTHREAD_MUTEX_INITIALIZER;
132: #endif
133:
134: //extern int errno;
135: #include <errno.h>
136: /*
137:     Description:
138:         Main entry point of the program. When started it is assumed that this is the only
139:         DM so it tries to hold an election. If the election fails then it goes into backup
140:         mode, otherwise normal DM mode.
141:
142:     Input:
143:         argc - The number of command line arguments
144:         argv - an array of the command line arguments
145:
146:     Output:
147:         -1 - the program has terminated unexpectedly
148:         1 - the program has terminated successfully
149:
150:     Changed:
151:         None
152:
153: */

```

```

154: int main(int argc, char** argv)
155: {
156: #ifndef WIN32
157:     sigset_t signal_set;
158: #endif
159:     SDM_TimeInit();
160:
161: #ifndef __VXWORKS__
162:     static struct option long_options[] = {
163:         {"help",0,0,'h'},
164:         {"debug",1,0,'g'},
165:         {"spacewire",1,0,'s'},
166:         {0,0,0,0}
167:     };
168:     int option_index;
169: #endif
170:     int option;
171:
172:     while(1)
173:     {
174: #ifdef __VXWORKS__
175:         option = getopt(argc,argv,"g:s:h:v");
176: #else
177:         option = getopt_long(argc,argv,"g:s:h:v", long_options, &option_index);
178: #endif
179:         if(option==-1)
180:             break; //no more options
181:         switch(option)
182:         {
183:             case 'h':
184:                 printf("Usage: dm [options] \n");
185:                 printf("Options: \n");
186:                 printf("--debug=<level> -g<level> \t \tSet level of debug messages \n \t \t \t \t \t \t0=none, 1=moderate, 2=verbose \n");
187:                 return 0;
188:             case 'g':
189:                 debug = atoi(optarg);
190:                 break;
191:             case 's':
192:                 spacewire = !!atoi(optarg);
193:                 break;

```



```

194:     case 'v':
195:         printf("SDM Version: %s Repo Rev: %i \n", SDM_VERSION, REVISION_NUMBER);
196:         return 0;
197:         case '?':
198:             break;
199:     }
200: }
201:
202: #ifdef WIN32 /*Windows will handle signals differently than the Linux implementation*/
203:     signal(SIGINT,SigHandler);
204:     sigset(SIGALRM,SigHandler);
205: #else /*Windows won't handle signal sets or a separate thread for signal handling*/
206:     //All subsequent threads block the SIGINT and SIGALRM signals so they aren't interrupted after
calling pthread_mutex_lock()
207:     //this avoids a deadlock situation by dedicating a single thread for signal handling.
208:     //These signal masks are inherited by all spawned threads
209:     sigemptyset(&signal_set);
210:     sigaddset(&signal_set, SIGALRM);
211:     sigaddset(&signal_set, SIGINT);
212:     pthread_sigmask(SIG_BLOCK, &signal_set, NULL);
213:     pthread_t signalHandler;
214:     if (0 != pthread_create(&signalHandler, NULL, &SigHandler, NULL))
215:     {
216:         perror("Could not start signal handler thread. \n");
217:         return -1;
218:     }
219:     pthread_detach(signalHandler);
220: #endif
221:
222: #ifdef SEND_IMA
223:     if (spacewire) {
224:         SendIMA(ImaDm, debug);
225:     }
226: #endif
227:
228:     // Get the Data Manager address
229:     if((Address_DM = GetNodeAddress(spacewire)) == 0)
230:     {
231:         printf("Unable to get the DM's IP address, using localhost instead. \n");
232:         Address_DM = L_LOCAL_HOST;
233:     }

```

```

234:     else
235:     {
236:         debug_f(3,"Data Manager address is 0x%lx \n",Address_DM);
237:     }
238:
239:     DataManager.setAddress(Address_DM);
240:     DataManager.setPort(PORT_DM);
241:     DataManager.setSensorID(DM_SENSOR_ID);
242:
243:     //TODO: Fix this for uclinux
244:     //CreatexTEDSDirectory();
245:
246:     // Start the main listener thread
247: #ifdef PNP_FAKE
248:     NetworkManager.setAddress(inet_addr("129.123.7.128"));
249:     NetworkManager.setPort(3605);
250:     NetworkManager.setSensorID(1);
251: #endif
252:
253: #ifdef PNP_BACKUP
254:     if (RunBackupListener() < 0)
255:         return -1;
256: #else
257:     pthread_t udpThread;
258:     pthread_attr_t attr;
259:     pthread_attr_init(&attr);
260:     pthread_attr_setstacksize(&attr,DM_STACK_SIZE);
261:     if (0 != pthread_create(&udpThread, &attr, &UdpListenerProcess, NULL))
262:     {
263:         perror ("Could not create UDPLListenerProcess thread: ");
264:         return -1;
265:     }
266:     pthread_join(udpThread,NULL);
267:     UdpListenerProcess(NULL);
268: #endif
269:     return 1;
270: }
271:
272:
273:
274: void* RegistrationHandler(void* args)

```

```

275: {
276:   SDMRegister registerMsg;
277:   SDMComponent_ID componentToRegister;
278:
279:   double endTime = 0;
280:   double timeOut = 5.0;
281:
282:   debug_f(1,"Queueing incoming registrations for 5 seconds... \n");
283:   sleep(5); //allow some registrations of both apps and devices to queue up
284:   registerNext = true;
285:   debug_f(1,"Now checking registration queues... \n");
286:   while(1)
287:   {
288:     if(GetCurTime() >= endTime) //Check for timeouts
289:     {
290:       registerNext = true;
291:     }
292:
293:     if(registerNext)
294:     {
295:       pthread_mutex_lock(&reg_queue_mutex);
296:       if(debug >= 2)
297:       {
298:         if(deviceQueue.size() > 0 || applicationQueue.size() > 0)
299:           debug_f(2,"Device Registration Queue: %i App Registration Queue: %li \n",
deviceQueue.size(), applicationQueue.size());
300:       }
301:
302:       if(deviceQueue.size() != 0)
303:       {
304:         componentToRegister = (SDMComponent_ID)deviceQueue.dequeue();
305:         if(componentToRegister.getPort() == PORT_SM || componentToRegister.getPort() ==
PORT_SPA1_MANAGER)
306:         {
307:           registerMsg.sensorIndex = componentToRegister.getSensorID();
308:           debug_f(2, "SDMRegister about to be sent to SM, sensorIndex: %i \n",
registerMsg.sensorIndex);
309:         }
310:         registerMsg.SendTo(componentToRegister);
311:
312:         char strTargetAddr[64];
313:         componentToRegister.IDToString(strTargetAddr, sizeof(strTargetAddr));

```

```

314:     debug_f(2,"SDMRegister sent to device at %s \n", strTargetAddr);
315:
316:     endTime = GetCurTime() + timeOut;
317:     registerNext = false;
318:     //sleep(1);
319: }
320: else if(applicationQueue.size() != 0)
321: {
322:     componentToRegister = applicationQueue.dequeue();
323:     if(componentToRegister.getSensorID() != 0)
324:     {
325:         registerMsg.sensorIndex = componentToRegister.getSensorID();
326:     }
327:     registerMsg.SendTo(componentToRegister);
328:
329:     char strTargetAddr[64];
330:     componentToRegister.IDToString(strTargetAddr, sizeof(strTargetAddr));
331:     debug_f(2,"SDMRegister sent to application at %s \n", strTargetAddr);
332:
333:     endTime = GetCurTime() + timeOut;
334:     registerNext = false;
335:     //sleep(1);
336: }
337: pthread_mutex_unlock(&reg_queue_mutex);
338: }
339: usleep(100000);
340: }
341:
342: return NULL;
343: }
344:
345:
346: void* CancelRegHandler(void* args)
347: {
348:     pthread_t CancelThread;
349:     pthread_attr_t attr;
350:     pthread_attr_init(&attr);
351:     pthread_attr_setstacksize(&attr,DM_STACK_SIZE);
352:     while(1)
353:     {
354:         pthread_mutex_lock(&cancel_queue_mutex);

```

```

355:   if(cancelNext && cancelxTEDSQueue.size() > 0)
356:   {
357:       debug_f(2, "Pulling off the cancelxTEDS queue... \n");
358:       cancelNext = false;
359:       xTEDSPParameters* toCancel = cancelxTEDSQueue.dequeue();
360:       pthread_mutex_unlock(&cancel_queue_mutex);
361:       if (0 != pthread_create(&CancelThread, &attr, CancelxTEDS, toCancel))
362:       {
363:           perror("Could not spawn CancelxTEDS thread. \n");
364:       }
365:       pthread_detach(CancelThread);
366:       usleep(10000);
367:   }
368:   else
369:   {
370:       pthread_mutex_unlock(&cancel_queue_mutex);
371:       sleep(1);
372:   }
373: }
374:
375: return NULL;
376: }
377:
378:
379: double GetCurTime()
380: {
381:   unsigned int seconds;
382:   unsigned int uSeconds;
383:   double curTime;
384:   SDM_GetTime(&seconds, &uSeconds);
385:   curTime = seconds + ((double)uSeconds/1000000.0);
386:   return curTime;
387: }
388:
389:
390: /*
391:   Description:
392:       Listener for UDP on DM_PORT. Registers the DM xTEDS. Calls the PipeInit to set up all
       necessary pipes
393:
394:   Input:

```

```

395:     None
396:
397: Output:
398:     None
399:
400: Changed:
401:     None
402:
403: */
404: void* UdpListenerProcess(void*)
405: {
406:     int iMessageSize = 0;
407:     char buf[BUFSIZE];    //Buffer for recieving incoming messages
408:     pthread_attr_t attr;
409:     pthread_t functionCallThread, tcpThread, xtedThread, registrationThread, cancelRegThread;
410:     int iStatus;
411:     int iSenderPort;
412:     struct sockaddr_in p;
413:     SDMComponent_ID SenderId;
414:     char SenderIdBuf[10];
415:     SDMComHandle ComHandle;
416:     SDMRegister registerMsg;
417:
418:     // Bind the UDP port and obtain socket descriptor
419:     udpSock = UDPPassive(PORT_DM);
420:     if (udpSock == IP_SOCKET_INVALID)
421:         return NULL;
422:
423:     PipeInit();
424:
425:     //Start the ChildFunctionCallProcess thread
426:     pthread_attr_init(&attr);
427:     pthread_attr_setstacksize(&attr,DM_STACK_SIZE);
428:     if (0 != pthread_create(&functionCallThread,&attr,ChildFunctionCallProcess,NULL))
429:     {
430:         perror("Could not start the child function call process thread. \n");
431:         return NULL;
432:     }
433:     pthread_detach(functionCallThread);
434:     if (0 != pthread_create(&tcpThread,&attr,TcpListenerProcess,NULL))
435:     {

```

```

436:     perror("Could not start the TCP listener thread. \n");
437:     return NULL;
438: }
439: pthread_detach(tcpThread);
440: if (0 != pthread_create(&registrationThread,&attr,RegistrationHandler,NULL))
441: {
442:     perror("Could not start the Registration Handle thread. \n");
443:     return NULL;
444: }
445: pthread_detach(tcpThread);
446: if (0 != pthread_create(&cancelRegThread,&attr,CancelRegHandler,NULL))
447: {
448:     perror("Could not start the Registration Handle thread. \n");
449:     return NULL;
450: }
451: pthread_detach(tcpThread);
452:
453:
454: // DM up and running with current version
455: printf ("DM SERVER %s running... \n",SDM_VERSION);
456: //
457: // Register the Data Manager's xTEDS
458: // Fill in the xTEDS parameters
459:
460:
461: xTEDSLibraryListNode* DMnode = MatchSID(DM_SENSOR_ID);
462: if(DMnode == NULL) //No DM previously registered
463: {
464:     SDMxTEDS xTEDSMessage;
465:     xTEDSMessage.source.setPort(DataManager.getPort());
466:     xTEDSMessage.source.setSensorID(DataManager.getSensorID());
467:     strncpy(xTEDSMessage.xTEDS, dmxDTED, sizeof(xTEDSMessage.xTEDS));
468:     iMessageSize = xTEDSMessage.Marshal(buf);
469:     SenderId.setAddress(Address_DM);
470:     xTEDS(new xTEDSParameters(buf,iMessageSize,SenderId, COM_HANDLE_INVALID));
471: }
472: else
473: {
474:     DMnode->data->setAddress(DataManager);
475:     DMnode->data->inUse->Signal();
476:     printf("Updating the registered DM Address to 0x%lx \n", DataManager.getAddress());

```

```

477: }
478:
479: // Initialize the IPC TCP socket with the function call process
480: usleep(1000);
481: int iAttemptCounter = 0;
482: bool bConnected = false;
483: while (!bConnected)
484: {
485:     if ((MsgTxPushSock = TCPconnect("127.0.0.1", PORT_DM_TEMP)) ==
IP_SOCKET_INVALID)
486:     {
487:         debug_f(3, "Error - UDPListenerProcess::Connection error with the function call process,
retrying. \n");
488:         iAttemptCounter++;
489:     }
490:     else
491:     {
492:         debug_f(3, "Connected to the function call process. \n");
493:         bConnected = true;
494:         break;
495:     }
496:
497:     if (iAttemptCounter >= 5)
498:     {
499:         debug_f(0, "UDPListenerProcess::Could not establish a connection with the child
function call process, quitting. \n");
500:         return NULL;
501:     }
502:     usleep(1000);
503: }
504: #ifdef PNP_BACKUP
505: xTEDSLibraryListNode* node;
506: // Tell the Task Manager to do a reset, if this is a newly elected DM
507: if(TaskManager.getPort() != 0)
508: {
509:     SDMCommand command;
510:     SDMComponent_ID temp_dmaddr;
511: // node = xTEDSList.head;
512: // while(node!=NULL)
513: // {
514: //     node->data->inUse->Wait();
515: //     if(node->data->getActive() == true)

```



```

516: //      {
517: //          if(node->data->getTargetPort() != PORT_DM      &&      node->data-
>getTargetPort() != PORT_PM      &&      node->data->getTargetPort() != PORT_SM      &&      node->data-
>getTargetPort() != PORT_TM)
518: //              {
519: //                  node->data->setActive(false);
520: //              }
521: //      }
522: //      node->data->inUse->Signal();
523: //      node = node->next;
524: //  }
525:      temp_dmaddr = DataManager;
526:      command.source = TaskManager;
527:      command.command_id.setMessage(5);
528:      command.command_id.setInterface(1);
529:      PUT_CHAR(command.data, MODE_RESET);
530:      temp_dmaddr.setPort(PORT_DM);
531:      temp_dmaddr.Marshal(command.data, 1);
532:      printf("Sending new DM address: 0x%lx to TM \n", temp_dmaddr.getAddress());
533:      command.length = HEADER_SIZE;
534:      command.Send(TaskManager);
535:  }
536: #endif
537: // The main message receiving loop
538: while(1)
539: {
540:     // Clear the buf so there are no residual messages in it
541:     iMessageSize = 0;
542:     // Receive the next message
543:     iStatus = UDPServ_recv(udpSock, buf, BUFSIZE);
544:     // Increment performance counters
545:     pthread_mutex_lock(&perf_counter_mutex);
546:     total_recd++;
547:     prevsec_recd++;
548:     pthread_mutex_unlock(&perf_counter_mutex);
549:     if(iStatus != -1 && iStatus != UDP_SERV_RECV_SHUTDOWN) //If no recieving error
process message
550:     {
551:         UDPgetip(&p);
552:         iMessageSize = iStatus;
553:         debug_f(4, "iMessageSize is %d \n", iMessageSize);
554:         iSenderPort = (int)ntohs(p.sin_port);

```

```

555:         SenderId.setAddress(p.sin_addr.s_addr);
556:         if (SenderId.getAddress() == L_LOCAL_HOST)
557:             SenderId.setAddress(Address_DM);
558:         SenderId.setPort(iSenderPort);
559:         ComHandle.Set(udpSock, false, &p);
560:
561:     if(buf[0] == SDM_Hello)
562:     {
563:         SDMHello helloMsg;
564:         if(helloMsg.Unmarshal(buf) < 0)
565:         {
566:             printf("Invalid SDMHello message! \n \n");
567:         }
568:
569:         SDMComponent_ID sm;
570:         SDMComponent_ID newReg = helloMsg.source;
571:         if ( newReg.getAddress() == 0 )
572:             newReg.setAddress(SenderId.getAddress());
573:
574:         char strTargetAddr[64];
575:         newReg.IDToString(strTargetAddr, sizeof(strTargetAddr));
576:         debug_f(2, "SDMHello from %s \n", strTargetAddr);
577:
578:         if(helloMsg.source.getPort() == PORT_SM)
579:         {
580:             sm = helloMsg.source;
581:             if ( sm.getAddress() == 0 )
582:                 sm.setAddress(SenderId.getAddress());
583:             sm.setPort(PORT_SM);
584:             debug_f(2, "SDMHello was from SM, sensorIndex: %i \n",
helloMsg.source.getSensorID());
585:         }
586:         switch(helloMsg.type)
587:         {
588:             case 'D': //Device
589:                 pthread_mutex_lock(&reg_queue_mutex);
590:                 if(!deviceQueue.find(newReg))
591:                 {
592:                     debug_f(2, "Device SDMHello received, adding to the queue \n");
593:                     deviceQueue.add(newReg);
594:                 }

```

```

595:         else
596:         {
597:             debug_f(2, "Ignoring duplicate SDMHHello \n");
598:         }
599:         pthread_mutex_unlock(&reg_queue_mutex);
600:
601:         if(helloMsg.source.getPort() != PORT_SM && helloMsg.source.getPort() !=
PORT_SPA1_MANAGER)
602:         {
603:             SendAckMessage(SDM_OK, true, newReg, ComHandle, true);
604:         }
605:         else
606:         {
607:             SendAckMessage(helloMsg.source.getSensorID(), true, sm, ComHandle, true);
608:         }
609:
610:         break;
611:     case 'A': //Application
612:         pthread_mutex_lock(&reg_queue_mutex);
613:         if(!applicationQueue.find(newReg))
614:         {
615:             debug_f(2, "Application SDMHHello received, adding to the queue \n");
616:             applicationQueue.add(newReg);
617:         }
618:         else
619:         {
620:             debug_f(2, "Ignoring duplicate SDMHHello \n");
621:         }
622:         pthread_mutex_unlock(&reg_queue_mutex);
623:
624:         if(helloMsg.source.getPort() != PORT_SM && helloMsg.source.getPort() !=
PORT_SPA1_MANAGER)
625:         {
626:             SendAckMessage(SDM_OK, true, newReg, ComHandle, true);
627:         }
628:         else
629:         {
630:             SendAckMessage(helloMsg.source.getSensorID(), true, sm, ComHandle, true);
631:         }
632:
633:         break;

```

```

634:         case 'C': //Core Component
635:             SendAckMessage(SDM_OK, true, newReg, ComHandle, true);
636:             registerMsg.SendTo(newReg);
637:             debug_f(2, "Ack and register msgs sent to core component \n");
638:             break;
639:         default:
640:             printf("Bad SDMHHello type with type: %c \n", helloMsg.type);
641:             break;
642:     }
643: }
644:     else if(buf[0] == SDM_xTEDS)
645:     {
646:         // Make sure we don't clog xTEDS registrations, if there are many outstanding,
647:         // drop this message
648:         if (addLibrary.GetWaitCount() < MAX_OUTSTANDING_XTEDS)
649:         {
650:             if (0 != pthread_create(&xtedThread,&attr,xTEDS,new
xTEDSParameters(buf,iMessageSize,SenderId,ComHandle)))
651:                 perror("Could not spawn xTEDS thread. \n");
652:
653:             pthread_detach(xtedThread);
654:         }
655:         else
656:         {
657:             debug_f(1, "xTEDS message dropped -- too many outstanding registrations. \n");
658:             droppedxTEDS++;
659:         }
660:     }
661:     else if(buf[0] == SDM_CancelxTEDS)
662:     {
663:         xTEDSParameters* cancelParams = new
xTEDSParameters(buf,iMessageSize,SenderId,ComHandle);
664:         pthread_mutex_lock(&cancel_queue_mutex);
665:         cancelxTEDSQueue.add(cancelParams);
666:         pthread_mutex_unlock(&cancel_queue_mutex);
667:         debug_f(2, "Queueing up a CancelxTEDS msg... \n");
668:     }
669:     else
670:     {
671:         SenderId.Marshal(SenderIdBuf,0);
672:         writeSem.Wait();

```

```

673:          TCPSend(MsgTxPushSock, SenderIdBuf, sizeof(SenderIdBuf));
674:          TCPSend(MsgTxPushSock, &iMessageSize, 4); //Write the size of the buf into the
push pipe
675:          TCPSend(MsgTxPushSock, buf, iMessageSize); //Write the Consume message into
the push pipe
676: #ifndef __VXWORKS__
677:          write(alert[1], &buf[0], 1); //Write the Consume byte into the notify pipe
678: #else
679:          write(alertPipe, &buf[0], 1);
680: #endif
681:          debug_f(4, "Alert written into pipe \n");
682:          writeSem.Signal();
683:      }
684:  }
685:  else
686:  {
687:      debug_f(4, "Error with UDPserv_rcv in DM main message loop. \n");
688:      // If udpSock is shutdown, UDPserv_rcv may return UDP_SERV_RECV_SHUTDOWN
before this thread can finish
689:      if (iStatus != UDP_SERV_RECV_SHUTDOWN)
690:          debug_f(4, "There was an error reading in the next message \n");
691:      usleep(1000); //Small sleep to prevent processor spinning
692:  }
693: }
694: return 0;
695: }
696:
697: /*
698:  Description:
699:      Listener for TCP on DM_PORT
700:
701:  Input:
702:      None
703:
704:  Output:
705:      None
706:
707:  Changed:
708:      None
709:
710: */
711: void* TcpListenerProcess(void*)

```

```

712: {
713:     int iSize = 0;
714:     char buf[LARGE_MSG_BUFSIZE];    //Buffer for recieving incoming messages
715:     int iStatus;
716:     int iCurLength;
717:     unsigned short usPort;
718:     struct sockaddr_in p;
719:     pthread_attr_t attr;
720:     pthread_t xtedThread;
721:     short sSdmLength;
722:     SDMComponent_ID SenderId;
723:     char SenderIdBuf[10];
724:     SDMComHandle ComHandle;
725:     SDMRegister registerMsg;
726:     bool bError = false;
727:
728:     pthread_attr_init(&attr);
729:     pthread_attr_setstacksize(&attr, DM_STACK_SIZE);
730:
731:     debug_f(3,"Listening for TCP message on port %hu \n", PORT_DM);
732:
733:     tcpListenSock = TCPpassive(PORT_DM, MAX_TCP_CONNECTIONS);
734:     if (tcpListenSock == IP_SOCKET_INVALID)
735:     {
736:         debug_f(0, "TCPListenerProcess:: Could not open listen socket. \n");
737:         return NULL;
738:     }
739:     //Start main loop
740:     while(1)
741:     {
742:         iSize = 0;
743:         tcpSock = TCPaccept(tcpListenSock, &p);
744:         if (tcpSock == IP_SOCKET_INVALID)
745:             continue;
746:         //
747:         // Make sure we at least grab the SDM header
748:         iCurLength = 0;
749:         bError = false;
750:         do
751:         {
752:             iStatus = TCPrecv(tcpSock, &buf[iCurLength], LARGE_MSG_BUFSIZE - iCurLength);

```

```

753:         if (iStatus < 0)
754:         {
755:             perror("TCPrecv:");
756:             bError = true;
757:             break;
758:         }
759:         else if (iStatus == 0)
760:         {
761:             bError = true;
762:             break;        // Peer has shutdown, don't handle partial messages
763:         }
764:         else
765:             iCurLength += iStatus;
766:     } while(iCurLength < HEADER_SIZE);
767:     if (bError)
768:     {
769:         TCPclose(tcpSock);
770:         continue;
771:     }
772:     //
773:     // Get the SDM message length
774:     sSdmLength = GET_SHORT(&buf[9]);
775:     //
776:     // Receive the rest of the message
777:     while(sSdmLength > iCurLength - HEADER_SIZE)
778:     {
779:         iStatus = TCPrecv(tcpSock, &buf[iCurLength], LARGE_MSG_BUFSIZE - iCurLength);
780:         if (iStatus < 0)
781:         {
782:             perror("TCPrecv:");
783:             bError = true;
784:             break;
785:         }
786:         else if (iStatus == 0)
787:         {
788:             bError = true;
789:             break;
790:         }
791:         else
792:             iCurLength += iStatus;
793:     }

```

```

794:     if (bError)
795:     {
796:         TCPclose(tcpSock);
797:         continue;
798:     }
799:     if(sSdmLength != iCurLength - HEADER_SIZE)
800:         printf("Error: More bytes recv'd than expected! \n");
801:         // Let the Unmarshal function report the error
802:
803:     pthread_mutex_lock(&perf_counter_mutex);
804:     total_recd++;
805:     prevsec_recd++;
806:     pthread_mutex_unlock(&perf_counter_mutex);
807:
808:     iSize = iCurLength;
809:     debug_f(4,"TCP: Size is %d \n", iSize);
810:     usPort = ntohs(p.sin_port);
811:     SenderId.setAddress(p.sin_addr.s_addr);
812:     if (SenderId.getAddress() == L_LOCAL_HOST)
813:         SenderId.setAddress(Address_DM);
814:     SenderId.setPort(usPort);
815:
816:     ComHandle.Set(tcpSock, true, &p);
817:
818:     if(buf[0] == SDM_Hello)
819:     {
820:         SDMHello helloMsg;
821:         if(helloMsg.Unmarshal(buf) < 0)
822:         {
823:             printf("Invalid SDMHello message! \n \n");
824:         }
825:
826:         SDMComponent_ID sm;
827:         SDMComponent_ID newReg = helloMsg.source;
828:         if ( newReg.getAddress() == 0 )
829:             newReg.setAddress(SenderId.getAddress());
830:
831:         char strTargetAddr[64];
832:         newReg.IDToString(strTargetAddr, sizeof(strTargetAddr));
833:         debug_f(2, "SDMHello from %s \n", strTargetAddr);
834:

```



```

835:     if(helloMsg.source.getPort() == PORT_SM)
836:     {
837:         sm = newReg;
838:         if ( sm.getAddress() == 0 )
839:             sm.setAddress(SenderId.getAddress());
840:         sm.setPort(PORT_SM);
841:         newReg.setSensorID(helloMsg.source.getSensorID());
842:         debug_f(2, "SDMHello was from SM, sensorIndex: %i \n", helloMsg.source.getSensorID());
843:     }
844:
845:     switch(helloMsg.type)
846:     {
847:         case 'D': //Device
848:             pthread_mutex_lock(&reg_queue_mutex);
849:             if(!deviceQueue.find(newReg))
850:             {
851:                 debug_f(2, "Device SDMHello received, adding to the queue \n");
852:                 deviceQueue.add(newReg);
853:             }
854:             else
855:             {
856:                 debug_f(2, "Ignoring duplicate SDMHello \n");
857:             }
858:             pthread_mutex_unlock(&reg_queue_mutex);
859:
860:             if(helloMsg.source.getPort() != PORT_SM && helloMsg.source.getPort() !=
PORT_SPA1_MANAGER)
861:             {
862:                 SendAckMessage(SDM_OK, true, newReg, ComHandle, true);
863:             }
864:             else
865:             {
866:                 SendAckMessage(helloMsg.source.getSensorID(), true, sm, ComHandle, true);
867:             }
868:
869:             break;
870:         case 'A': //Application
871:             pthread_mutex_lock(&reg_queue_mutex);
872:             if(!applicationQueue.find(newReg))
873:             {
874:                 debug_f(2, "Application SDMHello received, adding to the queue \n");

```

```

875:         applicationQueue.add(newReg);
876:     }
877:     else
878:     {
879:         debug_f(2, "Ignoring duplicate SDMHello \n");
880:     }
881:     pthread_mutex_unlock(&reg_queue_mutex);
882:
883:     if(helloMsg.source.getPort() != PORT_SM && helloMsg.source.getPort() !=
PORT_SPA1_MANAGER)
884:     {
885:         SendAckMessage(SDM_OK, true, newReg, ComHandle, true);
886:     }
887:     else
888:     {
889:         SendAckMessage(helloMsg.source.getSensorID(), true, sm, ComHandle, true);
890:     }
891:
892:     break;
893: case 'C': //Core Component
894:     SendAckMessage(SDM_OK, true, newReg, ComHandle, true);
895:     registerMsg.SendTo(newReg);
896:     debug_f(2, "Ack and register msgs sent to core component \n");
897:     break;
898: default:
899:     printf("Bad SDMHello type with type: %c \n", helloMsg.type);
900:     break;
901: }
902: }
903: else if(buf[0] == SDM_xTEDS)
904: {
905:     // Make sure we don't clog xTEDS registrations, if there are many outstanding,
906:     // drop this message
907:     if (addLibrary.GetWaitCount() < MAX_OUTSTANDING_XTEDS)
908:     {
909:         if (0 != pthread_create(&xtedThread,&attr,xTEDS,new
xTEDSPParameters(buf,iSize,SenderId,ComHandle)))
910:             perror("Could not spawn xTEDS thread. \n");
911:
912:         pthread_detach(xtedThread);
913:     }

```

```

914:         else
915:         {
916:             debug_f(1, "xTEDS message dropped -- too many outstanding registrations. \n");
917:             droppedxTEDS++;
918:         }
919:     }
920:     else if(buf[0] == SDM_CancelxTEDS)
921:     {
922:         xTEDSPParameters*          cancelParams          =          new
xTEDSPParameters(buf,iSize,SenderId,ComHandle);
923:         pthread_mutex_lock(&cancel_queue_mutex);
924:         cancelxTEDSQueue.add(cancelParams);
925:         pthread_mutex_unlock(&cancel_queue_mutex);
926:         debug_f(2, "Queueing up a CancelxTEDS msg... \n");
927:     }
928:     else
929:     {
930:         SenderId.Marshal(SenderIdBuf, 0);
931:         writeSem.Wait();
932:         TCPsend(MsgTxPushSock, SenderIdBuf, sizeof(SenderIdBuf)); //Write the ip address of
the reciever into the push pipe
933:         TCPsend(MsgTxPushSock, &iSize, 4); //Write the size of the buf into the push pipe
934:         TCPsend(MsgTxPushSock, buf, iSize); //Write the Consume message into the push pipe
935: #ifndef __VXWORKS__
936:         write(alert[1], &buf[0], 1); //Write the Consume byte into the notify pipe
937: #else
938:         write(alertPipe, &buf[0], 1);
939: #endif
940:         writeSem.Signal();
941:     }
942:     if(buf[0] != SDM_xTEDS && buf[0] != SDM_CancelxTEDS)
943:         TCPclose(tcpSock);
944: }
945: return 0;
946: }
947:
948: /*
949: Description:
950: Handling of the SDMConsume message. Forward a SDMSubreqst to the correct data
provider.
951:
952: Input:

```

```

953:     buf - the message in an array
954:     size - the length of the message in bytes
955:     SenderId - The sender address information
956:
957: Output:
958:     None
959:
960: Changed:
961:     None
962:
963: */
964: void Consume(char *buf, int size, const SDMComponent_ID& SenderId)
965: {
966:     SDMConsume msgConsumeRequest;
967:     SDMSubreqst msgSubRequest;
968:     xTEDSLibraryListNode* node;
969:
970:     if(msgConsumeRequest.Unmarshal(buf) < 0)
971:     {
972:         printf("Invalid SDMConsume message \n \n");
973:         return;
974:     }
975:     MessageReceived(&msgConsumeRequest);
976:
977: #ifdef PNP_BACKUP
978:     // Send the SDMConsume msg to all backup DMs
979:     msgConsumeRequest.destination.setAddress(SenderId.getAddress());
980:     pthread_mutex_lock(&dm_list_mutex);
981:     backupDMList.SendMessageToAll(msgConsumeRequest);
982:     pthread_mutex_unlock(&dm_list_mutex);
983: #endif
984:
985:     if (debug >= 1)
986:     {
987:         char strRequesterAddr[64];
988:         char strTargetAddr[64];
989:         char strTargetName[64];
990:         xTEDSList.GetDeviceName(msgConsumeRequest.source, strTargetName,
sizeof(strTargetName));
991:         msgConsumeRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
992:         msgConsumeRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));

```

```

993:     debug_f(1, "SDMConsume from %s for %s (%s) message 0x%x \n",
994:             strRequesterAddr, strTargetAddr, strTargetName,
995:             msgConsumeRequest.msg_id.getInterfaceMessagePair());
996: }
997: //debug_f(1,"command: Consume -- length %d port: %hu sensorID: %ld msgID: 0x%x
\n",size,msgConsumeRequest.destination.getPort(),msgConsumeRequest.source.getSensorID(),msgConsumeRequest.msg_id.getInterfaceMessagePair());
998:
999: //
1000: // Get the node structure for the sensor requested
1001: node = MatchSID(msgConsumeRequest.source.getSensorID());
1002: if(node==NULL) //If node is NULL reaches xsize there is no matching sensorID and request is
invalid
1003: {
1004:     printf("Could not find the sensor requested! \n");
1005:     //fflush(NULL);
1006:     return;
1007: }
1008: //
1009: // Put message together for the requested sensor
1010: msgSubRequest.destination = msgConsumeRequest.destination;
1011: msgSubRequest.destination.setAddress(SenderId.getAddress());
1012: msgSubRequest.source = msgConsumeRequest.source;
1013: msgSubRequest.msg_id = msgConsumeRequest.msg_id;
1014: //
1015: // If this is a request for the Data Manager, add a subscription table entry
1016: if(node->data->getSensorID() == DM_SENSOR_ID)
1017: {
1018:     node->data->inUse->Signal();
1019:     SDMComponent_ID SubscriberId;
1020:     SubscriberId.setAddress(SenderId.getAddress());
1021:     SubscriberId.setPort(msgSubRequest.destination.getPort());
1022:     // Add a subscription entry
1023:     Subscribe(SubscriberId, msgSubRequest.msg_id);
1024:     debug_f(1," \n");
1025:     return;
1026: }
1027: // Get the fault ID for the requested message
1028: msgSubRequest.fault_id = node->data->xtedsTree-
>getNotificationFaultMsgID(msgConsumeRequest.msg_id);
1029:
1030: if (debug >= 1)

```

```

1031:  {
1032:    char strTargetAddr[64];
1033:    char strRequesterAddr[64];
1034:    msgSubRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1035:    msgSubRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1036:    debug_f(1, " SDMSubreqst sent to %s from requester %s message 0x%x. \n",
1037:            strTargetAddr, strRequesterAddr,
1038:            msgSubRequest.msg_id.getInterfaceMessagePair());
1039:  }
1040:  //debug_f(1,"Msg sent to notification provider (0x%lx:%d SID=%ld) from 0x%lx:%d SID=%ld
Msg=0x%x          \n",          node->data->getAddress(),          msgSubRequest.source.getPort(),
msgSubRequest.source.getSensorID(),          msgSubRequest.destination.getAddress(),
msgSubRequest.destination.getPort(),          msgSubRequest.destination.getSensorID(),
msgSubRequest.msg_id.getInterfaceMessagePair());
1041:  //
1042:  // Increment the number of connections to the xTEDS
1043:  int n = node->data->getConnections();
1044:  n++;
1045:  node->data->setConnections(n);
1046:  node->data->inUse->Signal();
1047:  //
1048:  // If the sensor ID is filled in for the subscriber, it has an xTEDS and should be marked in
1049:  // the DM's provider subscription list so the DM can send out cancellations upon any type of
failure
1050:  if (msgSubRequest.destination.getSensorID() != 0 && msgSubRequest.destination.getSensorID()
!= 1)
1051:  {
1052:    debug_f(3, "Adding subscription to the g_SensorSubscriptions list... \n");
1053:    pthread_mutex_lock(&sensor_subs_mutex);
1054:    g_SensorSubscriptions.Add(msgSubRequest.source /*provider*/,
msgSubRequest.destination/*subscriber*/, msgSubRequest.msg_id);
1055:    pthread_mutex_unlock(&sensor_subs_mutex);
1056:  }
1057:  //
1058:  // Send the request
1059:  msgSubRequest.Send();
1060:  MessageSent(&msgSubRequest);
1061:  debug_f(1, " \n");
1062:  }
1063:  }
1064:
1065:  /*
1066:  Description:

```

```

1067:      Handling of the SDMCancel message. Forward a SDMDeletesub to the correct data
provider.
1068:
1069:  Input:
1070:      buf - the message in an array
1071:      size - the length of the message in bytes
1072:      SenderId - The sender address information
1073:
1074:  Output:
1075:      None
1076:
1077:  Changed:
1078:      None
1079:
1080: */
1081: void Cancel(char *buf, int size, const SDMComponent_ID& SenderId)
1082: {
1083:     SDMCancel msgCancelRequest;
1084:     SDMDeletesub msgDeleteSub;
1085:     xTEDSLibraryListNode* node;
1086:
1087:     if(msgCancelRequest.Unmarshal(buf) < 0)
1088:     {
1089:         printf("Invalid SDMCancel message! \n \n");
1090:         return;
1091:     }
1092:     MessageReceived(&msgCancelRequest);
1093:
1094:     if (debug >= 1)
1095:     {
1096:         char strRequesterAddr[64];
1097:         char strTargetAddr[64];
1098:         msgCancelRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1099:         msgCancelRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1100:         debug_f(1, "SDMCancel request from %s for %s message 0x%x \n",
1101:             strRequesterAddr, strTargetAddr,
1102:             msgCancelRequest.msg_id.getInterfaceMessagePair());
1103:     }
1104:
1105:     // Get the node structure
1106:     node = MatchSID(msgCancelRequest.source.getSensorID());

```

```

1107: if(node==NULL) //If n equals xsize there was no matching sensorID and request was invalid
1108: {
1109:     debug_f(3,"Could not find the sensor requested! \n");
1110:     return;
1111: }
1112:
1113: // This request's destination will be the sender of the request
1114: msgDeleteSub.destination = msgCancelRequest.destination;
1115: msgDeleteSub.destination.setAddress(SenderId.getAddress());
1116: msgDeleteSub.source = msgCancelRequest.source;
1117: msgDeleteSub.msg_id = msgCancelRequest.msg_id;
1118: // If this is a request for the Data Manager, add a subscription entry
1119: if(node->data->getSensorID() == DM_SENSOR_ID)
1120: {
1121:     node->data->inUse->Signal();
1122:     SDMComponent_ID SubscriberId;
1123:     SubscriberId.setAddress(SenderId.getAddress());
1124:     SubscriberId.setPort(msgDeleteSub.destination.getPort());
1125:     CancelSubscription(SubscriberId, msgDeleteSub.msg_id.getInterfaceMessagePair());
1126:     debug_f(1," \n");
1127:     return;
1128: }
1129:
1130: if (debug >= 1)
1131: {
1132:     char strRequesterAddr[64];
1133:     char strTargetAddr[64];
1134:     msgDeleteSub.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1135:     msgDeleteSub.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1136:     debug_f(1, " SDMDeletesub sent to %s from requester %s message 0x%x. \n",
1137:         strTargetAddr, strRequesterAddr,
1138:         msgDeleteSub.msg_id.getInterfaceMessagePair());
1139: }
1140:
1141: //
1142: // Decrement the number of connections to xTEDS
1143: int n = node->data->getConnections();
1144: n--;
1145: if(n < 0)
1146:     n = 0;
1147: node->data->setConnections(n);

```



```

1148: node->data->inUse->Signal();
1149: //
1150: // If the sensor ID is filled in for the subscriber, it has an xTEDS and should be removed from the
1151: // the DM's provider subscription list
1152: if (msgDeleteSub.destination.getSensorID() != 0 && msgDeleteSub.destination.getSensorID() !=
1153: 1)
1154: {
1155:     pthread_mutex_lock(&sensor_subs_mutex);
1156:     g_SensorSubscriptions.Delete(msgDeleteSub.destination /*subscriber*/,
1157: msgDeleteSub.msg_id);
1158:     pthread_mutex_unlock(&sensor_subs_mutex);
1159: }
1160: //
1161: // Send the delete subscription request message
1162: msgDeleteSub.Send();
1163: MessageSent(&msgDeleteSub);
1164:
1165: unsigned short usTPort = node->data->getTargetPort();
1166: if(n == 0 && usTPort != PORT_SM && usTPort != PORT_PM && usTPort != PORT_TM &&
1167: usTPort != PORT_SPA1_MANAGER) //If application has no subscribers send a kill message
1168: {
1169:     debug_f(3,"A process no longer has any subscribers and may need to be killed \n");
1170:     // FUTURE:
1171:     // This functionality still undefined
1172: }
1173: debug_f(1, " \n");
1174: }
1175:
1176: /*
1177: Description:
1178:     Handling of SDMCommand messages, forward it to the correct command provider.
1179:
1180: Input:
1181:     buf - the message in an array
1182:     size - the length of the message in bytes
1183:     SenderId - The address information of the requester
1184:
1185: Output:
1186:     None
1187:
1188: Changed:
1189:     None

```

```

1187:
1188: */
1189: void Command(char *buf, int size, const SDMComponent_ID& SenderId)
1190: {
1191:     SDMCommand msgCommand;
1192:
1193:     // Unmarshall the message
1194:     if (msgCommand.Unmarshal(buf) < 0)
1195:     {
1196:         printf("Invalid SDMCommand message! \n \n");
1197:         return;
1198:     }
1199:     MessageReceived(&msgCommand);
1200:
1201:     if (debug >= 1)
1202:     {
1203:         char strRequesterAddr[64];
1204:         char strTargetAddr[64];
1205:         msgCommand.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1206:         msgCommand.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1207:         debug_f(1, "SDMCommand from %s for %s message 0x%x \n",
1208:             strRequesterAddr, strTargetAddr,
1209:             msgCommand.command_id.getInterfaceMessagePair());
1210:     }
1211:     //
1212:     // Get the sensor id and pull the xTEDS structure
1213:     unsigned long RequestedSensorId = msgCommand.source.getSensorID();
1214:     xTEDSLibraryListNode* node = MatchSID(RequestedSensorId);
1215:     if(node==NULL)    //If sensor ID did not match
1216:     {
1217:         printf("Sensor ID did not match any available \n \n");
1218:         return;
1219:     }
1220:     //
1221:     // Get the command id and make sure it is valid for this data provider
1222:     SDMMMessage_ID RequestedCommandId (msgCommand.command_id);
1223:     if (!node->data->xtedsTree->isCommandIdValid(RequestedCommandId))
1224:     {
1225:         printf("Invalid message id for command request. \n");
1226:         node->data->inUse->Signal();
1227:         return;

```

```

1228: }
1229:
1230: SDMMMessage_ID CommandFaultId (node->data->xtedsTree-
>getCommandFaultMsgID(RequestedCommandId));
1231: msgCommand.fault_id = CommandFaultId;
1232: //
1233: // If this is a command for the Data Manager
1234: if (node->data->getSensorID() == DM_SENSOR_ID)
1235: {
1236:     if (RequestedCommandId == CMD_ENABLE_LOGGING) //Enable logging command
1237:     {
1238:         debug_f(1, " DM command to enable log messages received. \n");
1239:         pthread_mutex_lock(&log_service_mutex);
1240:         if (log_service.NeedsInit())
1241:             log_service.SetLogFile("Data Manager Message Log \n", "dmmessages.log");
1242:         log_service.AddMessageType(&msgCommand);
1243:         pthread_mutex_unlock(&log_service_mutex);
1244:         node->data->inUse->Signal();
1245:         return;
1246:     }
1247:     else if (RequestedCommandId == CMD_DISABLE_LOGGING)//Disable logging command
1248:     {
1249:         debug_f(1, " DM command to disable log messages received. \n");
1250:         pthread_mutex_lock(&log_service_mutex);
1251:         log_service.RemoveMessageType(&msgCommand);
1252:         pthread_mutex_unlock(&log_service_mutex);
1253:         node->data->inUse->Signal();
1254:         return;
1255:     }
1256:     else
1257:     {
1258:         printf("Received invalid command of 0x%x for DataManager \n",
RequestedCommandId.getInterfaceMessagePair());
1259:         node->data->inUse->Signal();
1260:         return;
1261:     }
1262: }
1263: //
1264: // Debug output
1265: if (debug >= 1)
1266: {

```

```

1267:   char strRequesterAddr[64];
1268:   char strTargetAddr[64];
1269:   msgCommand.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1270:   msgCommand.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1271:   debug_f(1, " SDMCommand sent to %s requested from %s message 0x%x \n",
1272:         strTargetAddr, strRequesterAddr,
1273:         msgCommand.command_id.getInterfaceMessagePair());
1274:   }
1275:
1276:   if (debug >= 3)
1277:   {
1278:       debug_f(3, " Data: ");
1279:       for (int i = 0; i < msgCommand.length; i++)
1280:           debug_f(3, "%x ", msgCommand.data[i]);
1281:   }
1282:   debug_f(3, " \n");
1283:   //
1284:   // Send the message to the provider
1285:   SDMComponent_ID ProviderId;
1286:   ProviderId.setAddress(node->data->getAddress());
1287:   ProviderId.setPort((short)node->data->getTargetPort());
1288:   msgCommand.Send(ProviderId);
1289:   MessageSent(&msgCommand);
1290:
1291:   node->data->inUse->Signal();
1292:   debug_f(1, " \n");
1293: }
1294:
1295: /*
1296:  Handle service requests to the Data Manager services only. All other service requests for
1297:  other devices should use the SDMService interface.
1298:
1299: */
1300: void Serreqst(const char* buf, int size, const SDMComponent_ID& SenderId)
1301: {
1302:   SDMSerreqst msgServiceRequest;
1303:   if (msgServiceRequest.Unmarshal(buf) < 0)
1304:   {
1305:       printf("Invalid SDMSerreqst message. \n");
1306:       return;
1307:   }

```

```

1308:
1309: const unsigned long REQUESTED_SENSOR_ID = msgServiceRequest.source.getSensorID();
1310:
1311: if (debug >= 1)
1312: {
1313:     char strRequesterAddr[64];
1314:     char strTargetAddr[64];
1315:     char strRequesterName[64];
1316:     xTEDSList.GetDeviceName(msgServiceRequest.destination, strRequesterName,
sizeof(strRequesterName));
1317:     msgServiceRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1318:     msgServiceRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1319:     debug_f(1, "SDMSerreqst from %s (%s) for DataManager message 0x%x \n",
1320:         strRequesterAddr, strRequesterName,
1321:         msgServiceRequest.command_id.getInterfaceMessagePair());
1322: }
1323:
1324: //
1325: // Only handle SDMSerreqst messages for the DM's services
1326: if (REQUESTED_SENSOR_ID != DM_SENSOR_ID)
1327: {
1328:     printf("Error: Serreqst is for a device other than the Data Manager. \n");
1329:     return;
1330: }
1331: //
1332: // Get the DM's xTEDS library
1333: xTEDSLibraryListNode* pDmLibraryNode = MatchSID(DM_SENSOR_ID);
1334: if (pDmLibraryNode == NULL)
1335: {
1336:     // This should never happen, but assure we're not using a null pointer
1337:     printf("Fatal Error: DM xTEDS library could not be found. \n");
1338:     return;
1339: }
1340:
1341: const SDMMessage_ID idRequestedCommand(msgServiceRequest.command_id);
1342: const SDMMessage_ID idDataReply(pDmLibraryNode->data->xtedsTree-
>getServiceDataMsgID(idRequestedCommand));
1343: pDmLibraryNode->data->inUse->Signal();
1344:
1345: if(idRequestedCommand == SER_SEND_SENSOR_ID || idRequestedCommand ==
SER_SEND_PID

```

```

1346:    || idRequestedCommand == SER_SID_TO_SPANODE || idRequestedCommand ==
SER_SID_TO_IP
1347:    || idRequestedCommand == SER_COMPID_TO_COMPKEY)
1348:  {
1349:    //
1350:    // Handle the request if the command id is valid
1351:    unsigned long ulRequestedSensorId = GET_ULONG(&msgServiceRequest.data[0]);
1352:    SDMComponent_ID RequesterId;
1353:    RequesterId.setAddress(SenderId.getAddress());
1354:    RequesterId.setPort(msgServiceRequest.destination.getPort());
1355:    ServicePublish(RequesterId, ulRequestedSensorId, idDataReply,
(int)msgServiceRequest.seq_num);
1356:  }
1357:  else if ( idRequestedCommand == SER_GET_ERRORS )
1358:  {
1359:    SDMComponent_ID RequesterId;
1360:    RequesterId.setAddress(SenderId.getAddress());
1361:    RequesterId.setPort(msgServiceRequest.destination.getPort());
1362:    ServicePublish(RequesterId, 0, idDataReply, (int)msgServiceRequest.seq_num);
1363:  }
1364:  else
1365:    printf("Command ID 0x%x does not match any Data Manager service. \n",
idRequestedCommand.getInterfaceMessagePair());
1366:
1367:  debug_f(1, " \n");
1368: }
1369: /*
1370:  Description:
1371:    Handling of SDMSERVICE messages, forward a SDMSerreqst to the correct service provider.
1372:
1373:  Input:
1374:    buf - the message in an array
1375:    size - the length of the message in bytes
1376:    SenderId - The address information of the requester
1377:
1378:  Output:
1379:    None
1380:
1381:  Changed:
1382:    None
1383:
1384:  */

```

```

1385: void Service(char *buf, int size, const SDMComponent_ID& SenderId)
1386: {
1387:     SDMService msgService;           // Incoming request message
1388:     SDMSerreqst msgRequest;          // Outgoing request message
1389:
1390:     if(msgService.Unmarshal(buf) < 0) //Check to see that the service message is at least the
minimum correct number of bytes
1391:     {
1392:         printf("Invalid SDMService message \n \n");
1393:         return;
1394:     }
1395:     MessageReceived(&msgService);
1396:
1397:     if (debug >= 1)
1398:     {
1399:         char strRequesterAddr[64];
1400:         char strTargetAddr[64];
1401:         msgService.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1402:         msgService.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1403:         debug_f(1, "SDMService from %s for %s message 0x%x \n",
1404:             strRequesterAddr, strTargetAddr,
1405:             msgService.command_id.getInterfaceMessagePair());
1406:     }
1407:
1408:     //
1409:     // Get the sensor id and pull the xTEDS structure
1410:     unsigned long RequestedSensorId = msgService.source.getSensorID();
1411:     xTEDSLibraryListNode* node = MatchSID(RequestedSensorId);
1412:     if(node==NULL)
1413:     {
1414:         printf("Sensor ID did not match any available \n \n");
1415:         return;
1416:     }
1417:     //
1418:     // Get the requested command id and make sure it is a service in the provider's xTEDS
1419:     SDMMMessage_ID RequestedCommandID (msgService.command_id;
1420:     SDMMMessage_ID SerReplyID (node->data->xtedsTree-
>getServiceDataMsgID(RequestedCommandID));
1421:     if(SerReplyID == INVALID_MESSAGE_ID)
1422:     {
1423:         printf("CommandID 0x%x is not part of a Request
\n",RequestedCommandID.getInterfaceMessagePair());

```

```

1424:     node->data->inUse->Signal();
1425:     return;
1426: }
1427: SDMMMessage_ID          SerFaultID          (node->data->xtedsTree-
>getServiceFaultMsgID(RequestedCommandID));
1428: //
1429: // If this is a request for the Data Manager's services
1430: if(node->data->getSensorID() == DM_SENSOR_ID)
1431: {
1432:     if(RequestedCommandID == SER_SEND_SENSOR_ID || RequestedCommandID ==
SER_SEND_PID
1433:         || RequestedCommandID == SER_SID_TO_SPANODE || RequestedCommandID ==
SER_SID_TO_IP
1434:         || RequestedCommandID == SER_COMPID_TO_COMPKEY)
1435:     {
1436:         RequestedSensorId = GET_LONG(&msgService.data[0]);
1437:         node->data->inUse->Signal();
1438:         SDMComponent_ID RequesterId;
1439:         RequesterId.setAddress(SenderId.getAddress());
1440:         RequesterId.setPort(msgService.destination.getPort());
1441:         ServicePublish(RequesterId,          RequestedSensorId,          SerReplyID,
(int)msgService.seq_num);
1442:         debug_f(1," \n");
1443:         return;
1444:     }
1445:     else if ( RequestedCommandID == SER_GET_ERRORS )
1446:     {
1447:         SDMComponent_ID RequesterId;
1448:         RequesterId.setAddress(SenderId.getAddress());
1449:         RequesterId.setPort(msgService.destination.getPort());
1450:         ServicePublish(RequesterId, 0, SerReplyID, (int)msgService.seq_num);
1451:     }
1452:     else
1453:     {
1454:         printf("Invalid Command ID of 0x%x for data manager service request \n",
RequestedCommandID.getInterfaceMessagePair());
1455:         node->data->inUse->Signal();
1456:         return;
1457:     }
1458: }
1459: // Assure the requested service is valid for this producer
1460: if (!node->data->xtedsTree->isServiceIdValid(RequestedCommandID))

```



```

1461: {
1462:     printf("Invalid message id for service request. \n");
1463:     node->data->inUse->Signal();
1464:     return;
1465: }
1466:
1467: // Copy the address of the requester
1468: msgRequest.destination.setAddress(SenderId.getAddress());
1469: msgRequest.source = msgService.source;
1470:
1471: // Copy the command, reply, and fault message ids into the request
1472: msgRequest.command_id = RequestedCommandID;
1473: msgRequest.reply_id = SerReplyID;
1474: msgRequest.fault_id = SerFaultID;
1475:
1476: // Copy requester info into the request
1477: msgRequest.destination.setPort(msgService.destination.getPort());
1478: msgRequest.destination.setSensorID(msgService.destination.getSensorID());
1479:
1480: // Copy the sequence number into the request
1481: msgRequest.seq_num = msgService.seq_num;
1482:
1483: // Copy the data portion and length into the request
1484: memcpy(&msgRequest.data[0], &msgService.data[0], msgService.length);
1485: msgRequest.length = msgService.length;
1486: //
1487: // Debug output
1488: if (debug >= 1)
1489: {
1490:     char strRequesterAddr[64];
1491:     char strTargetAddr[64];
1492:     msgRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1493:     msgRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1494:     debug_f(1, " Sent SDMSerreqst to %s from %s message 0x%x length %d \n",
1495:         strTargetAddr, strRequesterAddr,
1496:         msgRequest.command_id.getInterfaceMessagePair(), msgRequest.length);
1497: }
1498:
1499: if (debug >= 3)
1500: {
1501:     debug_f(3, " data: ");

```

```

1502:     for(int i = 0; i < msgRequest.length; i++)
1503:         debug_f(3,"%x ", msgRequest.data[i]);
1504: }
1505: debug_f(3," \n");
1506: //
1507: // Send the request message to the provider
1508: SDMComponent_ID ProviderId;
1509: ProviderId.setAddress(node->data->getAddress());
1510: ProviderId.setPort(node->data->getTargetPort());
1511: msgRequest.Send(ProviderId);
1512: MessageSent(&msgRequest);
1513:
1514: node->data->inUse->Signal();
1515: debug_f(1, " \n");
1516: }
1517:
1518: /*
1519: Description:
1520:     Handling of the SDMReqReg messages. There are two forms of ReqReg calls, one to
perform an exhaustive search
1521:     of all registered xTEDS as a request of an application, and one to perform a search of only
newly registered
1522:     or unregistered xTEDS. These cases are handled when xTEDref is -1 (exhaustive search) or
the index numbered
1523:     value of the xTEDS from which to begin.
1524:
1525: Input:
1526:     MsgBuf - the message in an array
1527:     MsgSize - the length of the message in bytes
1528:     subref - a pointer to a subscription list node
1529:     xTEDref - an index reference to the xTEDS node, this will be -1 if all xTEDS to be searched
for matches
1530:     if this index is >= 0, search only the xTEDS registered in that slot. In this case,
RequesterIPAddr
1531:     and RequesterPort are the address and port of the subscriber.
1532:     RequesterIPAddr - the ip address in dot notation
1533:     RequesterPort - the port to respond on if SDMReqReg.destination.port == 0 &&
SDMReqReg.reply == SDM_CURRENT
1534:     xTEDSAction - One of REGISTRATION_MODIFICATION,
DEREGISTRATION_MODIFICATION, or UPDATE_MODIFICATION when
1535:     the ReqReg is being performed on a subscription to fill the "type" in a SDMRegInfo, or
MOD_NOT_APPLICABLE otherwise
1536:

```

```

1537: Output:
1538:     None
1539:
1540: Changed:
1541:     None
1542:
1543: */
1544: void ReqReg(char *MsgBuf, int MsgSize, const int xTEDref, const SDMComponent_ID&
SenderId, int xTEDSAction) //Search xTEDS for matches of item_name and quallist and return
SDMRegInfo
1545: {
1546:     unsigned int RegInfoCountSent=0;
1547:     char* MessageDefString; //Pointer to message definitions to send
1548:     char* xTEDSSection; //Pointer to xTEDS portion to send
1549:     MessageDef* msgdefs = NULL; //This is a heap-allocated object, this method must
delete it
1550:     MessageDef* cur_msgdef;
1551:     unsigned short usPort = 0;
1552:     bool result = false;
1553:     bool ContinueSearch = true;
1554:     SDMReqReg msgRequest;
1555:     SDMRegInfo msgInfo;
1556:     xTEDSLibraryListNode* node;
1557:     int iRequestType = 0;
1558:     SDMComponent_ID id; //The component ID of the requesting application
1559:     long dev_length = 0, interface_length = 0;
1560:
1561:     //Copy from MsgBuf into local variable for ease of use
1562:     if(msgRequest.Unmarshal(MsgBuf) < 0)
1563:     {
1564:         printf("Invalid SDMReqReg message \n \n");
1565:         return;
1566:     }
1567:     MessageReceived(&msgRequest);
1568:
1569:     //Save the requester information in "id"
1570:     usPort = msgRequest.destination.getPort();
1571:     if(msgRequest.reply == SDM_REQREG_CURRENT && usPort == 0)
1572:         usPort = SenderId.getPort();
1573:     if(msgRequest.destination.getAddress() == 0)
1574:         id.setAddress(SenderId.getAddress());
1575:     else

```

```

1576:     id.setAddress(msgRequest.destination.getAddress());
1577: id.setPort(usPort);
1578:
1579: if (debug >= 1)
1580: {
1581:     char strRequesterAddr[64];
1582:     char strTargetAddr[64];
1583:     msgRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
1584:     msgRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1585:     debug_f(1, "SDMReqReg from %s for %s %d device: \"%s\" item_name: \"%s\" \n",
1586:         strRequesterAddr, strTargetAddr, msgRequest.id, msgRequest.device,
1587:         msgRequest.item_name);
1588: }
1589: //If the reply style isn't SDM_REQREG_CURRENT (i.e. a subscription needs to be entered or
1590: //cancelled) and this isn't already
1591: //a previous request corresponding to an xTEDS
1592: if(msgRequest.reply != SDM_REQREG_CURRENT && xTEDref == -1) //A subscription is
1593: //wanted or needs to be canceled
1594: {
1595:     SDMComponent_ID SubscriberId;
1596:     SubscriberId.setAddress(SenderId.getAddress());
1597:     SubscriberId.setPort(usPort);
1598:     pthread_mutex_lock(&subscription_mutex);
1599:     result = subscribers.addOrRemoveSubscription(msgRequest.reply, SubscriberId,
1600:         msgRequest.source, msgRequest.device, msgRequest.interface, msgRequest.item_name,
1601:         msgRequest.quallist, msgRequest.id, REQ_REG_FUTURE, debug);
1602:     pthread_mutex_unlock(&subscription_mutex);
1603:     if(result == false)
1604:         return;
1605: #ifdef PNP_BACKUP
1606:     // Send the ReqReg subscription to all backup DMs
1607:     pthread_mutex_lock(&dm_list_mutex);
1608:     msgRequest.destination = SubscriberId;
1609:     backupDMList.SendMessageToAll(msgRequest);
1610:     pthread_mutex_unlock(&dm_list_mutex);
1611: #endif
1612: }
1613:
1614: //Count the number of xTEDS currently registered
1615: int numxTEDS = 0;
1616: node = xTEDSList.head;

```

```

1613: while(node!=NULL)
1614: {
1615:     numxTEDS++;
1616:     node = node->next;
1617: }
1618:
1619: int *SensorMatches = new int[numxTEDS];    // Holds a RegInfo match count for each
currently registered xTEDS
1620: const int SensorMatchesSize = numxTEDS;    // Number of currently registered xTEDS
1621:
1622: for(int i = 0; i < SensorMatchesSize; i++)    // Initialize the SensorMatches array
1623:     SensorMatches[i] = 0;
1624:
1625: numxTEDS = 0;
1626: node = xTEDSList.head;
1627: debug_f(3,"Starting search for matching in xTEDS \n");
1628:
1629: //If the item name is empty, match only on qualifiers/device/interface
1630: //Type 0 has item_name, iRequestType 1 does not have item_name, iRequestType 2 is a regular
expression for the item_name
1631: if(msgRequest.item_name[0] == '\0')
1632: {
1633:     iRequestType = REQTYPE_EMPTYITEM;
1634: }
1635: else
1636: {
1637:     //Determine whether this is a regular expressions-style ReqReg
1638:     iRequestType = ParseItemName(msgRequest.item_name,debug);
1639: }
1640: //Traverse the xTEDS library to perform a RegInfo
1641: while(node!=NULL && ContinueSearch == true)
1642: {
1643:     if(node->data->inUse->Wait() == 0)
1644:     {
1645:         //Try to match only xTEDS whose component ID matches the request, or if the request's
1646:         //sensorID is zero, try to match all xTEDS
1647:         if(node->data->getComponentID()==msgRequest.source                ||
msgRequest.source.getSensorID() == 0)
1648:         {
1649:             if (node->data->getAvailable())
1650:             {
1651:                 // If this node is not used by any xTEDS, skip over it

```

```

1652:                // (available means it's not being used)
1653:                node->data->inUse->Signal();
1654:                node = node->next;
1655:                continue;
1656:            }
1657:            debug_f(4, "Trying ReqReg on device %s \n", node->data->getName());
1658:            dev_length = strlen(msgRequest.device);
1659:            interface_length = strlen(msgRequest.interface);
1660:            switch(iRequestType)
1661:            {
1662:                case REQTYPE_USEITEM:    //If this is a RegInfo related to a specific item
name
1663:                    if(dev_length > 0 && interface_length > 0)
1664:                        msgdefs = node->data->xtedsTree-
>RegInfo(msgRequest.item_name, msgRequest.quallist, msgRequest.device, msgRequest.interface);
1665:                    else if(dev_length == 0 && interface_length == 0)
1666:                        msgdefs = node->data->xtedsTree-
>RegInfo(msgRequest.item_name, msgRequest.quallist, NULL, NULL);
1667:                    else if(dev_length == 0)
1668:                        msgdefs = node->data->xtedsTree-
>RegInfo(msgRequest.item_name, msgRequest.quallist, NULL, msgRequest.interface);
1669:                    else
1670:                        msgdefs = node->data->xtedsTree-
>RegInfo(msgRequest.item_name, msgRequest.quallist, msgRequest.device, NULL);
1671:                    break;
1672:                case REQTYPE_EMPTYITEM://If this is a RegInfo not related to a specific
item name
1673:                    if(dev_length > 0 && interface_length > 0)
1674:                        msgdefs = node->data->xtedsTree-
>AllRegInfo(msgRequest.quallist, msgRequest.device, msgRequest.interface);
1675:                    else if(dev_length == 0 && interface_length == 0)
1676:                        msgdefs = node->data->xtedsTree-
>AllRegInfo(msgRequest.quallist, NULL, NULL);
1677:                    else if(dev_length == 0)
1678:                        msgdefs = node->data->xtedsTree-
>AllRegInfo(msgRequest.quallist, NULL, msgRequest.interface);
1679:                    else
1680:                        msgdefs = node->data->xtedsTree-
>AllRegInfo(msgRequest.quallist, msgRequest.device, NULL);
1681:                    break;
1682:                case REQTYPE_REGEX:    //If this is a regular expressions style RegInfo
1683:                    try
1684:                    {

```

```

1685:                if(dev_length > 0 && interface_length > 0)
1686:                    msgdefs = node->data->xtedsTree->
>RegexRegInfo(msgRequest.item_name,msgRequest.quallist,msgRequest.device,msgRequest.interface);
1687:                else if(dev_length == 0 && interface_length == 0)
1688:                    msgdefs = node->data->xtedsTree->
>RegexRegInfo(msgRequest.item_name,msgRequest.quallist,NULL,NULL);
1689:                else if(dev_length == 0)
1690:                    msgdefs = node->data->xtedsTree->
>RegexRegInfo(msgRequest.item_name,msgRequest.quallist,NULL,msgRequest.interface);
1691:                else
1692:                    msgdefs = node->data->xtedsTree->
>RegexRegInfo(msgRequest.item_name,msgRequest.quallist,msgRequest.device,NULL);
1693:            }
1694:            catch(...)
1695:            {
1696:                ContinueSearch = false;
1697:                printf("Error: Invalid Regular Expression \n");
1698:                //fflush(NULL);
1699:            }
1700:            break;
1701:        default:
1702:            break;
1703:    }
1704:    //If for some reason, the search failed or produced an error, stop
1705:    if(ContinueSearch == false)
1706:    {
1707:        node->data->inUse->Signal();
1708:        break;
1709:    }
1710:    cur_msgdef = msgdefs;
1711:    //Put together SDMRegInfo messages
1712:    while(cur_msgdef!=NULL)
1713:    {
1714:        // Increase the number of matches to this sensor
1715:        // Do a bounds check, a device could have registered
1716:        if (numxTEDS < SensorMatchesSize)
1717:            SensorMatches[numxTEDS]++;
1718:
1719:        if(node->data->getActive() == true || xTEDSAction ==
DEREGISTRATION_MODIFICATION)
1720:        {

```

```

1721:          msgInfo.id = msgRequest.id;          //Copy in the request stream ID
from the ReqReg
1722:          if(xTEDSAction != DEREGISTRATION_MODIFICATION || xTEDref == -
1)
1723:              msgInfo.type = SDM_REGINFO_REGISTRATION;
1724:          else
1725:              msgInfo.type = SDM_REGINFO_CANCELLATION;
1726:          //
1727:          //Set the source address to be the xTEDS that matched
1728:          msgInfo.source.setSensorID(node->data->getSensorID());
1729:          debug_f(4,"Inserted SensorID of %ld \n",node->data->getSensorID());
1730:
1731:          msgInfo.source.setPort(node->data->getTargetPort());
1732:          debug_f(4,"Inserted Port of %hu \n",node->data->getTargetPort());
1733:
1734:          msgInfo.source.setAddress(node->data->getAddress());
1735:          debug_f(4,"Inserted Address of 0x%x \n",node->data->getAddress());
1736:
1737:          //
1738:          //Set the message ID to the message that matched
1739:          msgInfo.msg_id = cur_msgdef->GetInterfaceMessageID();
1740:          debug_f(4,"Inserted          MessageID          of          0x%x
\n",msgInfo.msg_id.getInterfaceMessagePair());
1741:
1742:          debug_f(5,"xTEDref is %d \n",xTEDref);
1743:          //Save a reference to both the message def. and the xTEDS portion
1744:          MessageDefString = cur_msgdef->GetDefinitions();
1745:          xTEDSSection = cur_msgdef->GetxTEDSPortion();
1746:          if((debug >= 1 && xTEDref == -1) || (debug >=1 && xTEDref ==
numxTEDS)) //Print the message being sent
1747:          {
1748:              if(strlen(MessageDefString) > 15)
1749:              {
1750:                  char strTargetAddr[64];
1751:                  char strDeviceAddr[64];
1752:                  SenderId.IDToString(strTargetAddr, sizeof(strTargetAddr));
1753:                  node->data->getComponentID().IDToString(strDeviceAddr,
sizeof(strDeviceAddr));
1754:                  debug_f(1,"  SDMRegInfo match for %s (%s), message sent to %s \n",
node->data->getName(), strDeviceAddr, strTargetAddr);
1755:                  debug_f(2,"  msg_def: %s xTEDS Portion %s \n", MessageDefString,
xTEDSSection);

```



```

1756:         }
1757:     }
1758:     //Copy in the message definition portion, including null terminator
1759:     memcpy(msgInfo.msg_def,MessageDefString,strlen(MessageDefString)+1);
1760:     msgInfo.msg_def[strlen(MessageDefString)] = '\0';
1761:     //Copy in the xTEDS portion corresponding with the message, including null
terminator
1762:     memcpy(msgInfo.xTEDS_section,xTEDSSection,strlen(xTEDSSection)+1);
1763:     msgInfo.xTEDS_section[strlen(xTEDSSection)] = '\0';
1764:     //
1765:     //Send RegInfo's if a message definitions was matched
1766:     if(xTEDref == -1 && strlen(MessageDefString) > 15) //If reply is wanted
and subscription is entered
1767:     {
1768:         msgInfo.Send(id);
1769:         MessageSent(&msgInfo);
1770:         RegInfoCountSent++;
1771:         debug_f(4,"Sent a reply to SDMReqReg to 0x%lx at port %hu
\n",SenderId.getAddress(),usPort);
1772:     }
1773:     //If this is a subscription match, and a message def matched for a recent
registration
1774:     else if(numxTEDS == xTEDref && strlen(MessageDefString) > 15)
1775:     {
1776:         id = SenderId;
1777:         msgInfo.Send(id);
1778:         MessageSent(&msgInfo);
1779:         RegInfoCountSent++;
1780:         debug_f(4,"Sent a reply to SDMReqReg to 0x%lx at port %hu for
subscription \n", SenderId.getAddress(), SenderId.getPort());
1781:     }
1782:     }
1783:     cur_msgdef = cur_msgdef->next;
1784:     }
1785:     if(msgdefs!=NULL)
1786:     {
1787:         delete msgdefs;
1788:         msgdefs = NULL;
1789:     }
1790:     }
1791:     node->data->inUse->Signal();
1792:     }

```

```

1793:     node = node->next;
1794:
1795:     // If xTED is not equal to -1 then this is a subscription looking for new registrations
1796:     if(xTEDref != -1 && xTEDref == numxTEDS)
1797:     {
1798:         // Check to see if the new registration has the item_name and qual_list
1799:         if(numxTEDS < SensorMatchesSize && SensorMatches[numxTEDS] == 0)
1800:         {
1801:             delete [] SensorMatches;
1802:             // If no new matching info function does not need to finish
1803:             return;
1804:         }
1805:     }
1806:
1807:     numxTEDS++;
1808: }
1809: //If there were no RegInfo's sent, check to see if an inactive xTEDS will match according to the
value in sensornames
1810: if(RegInfoCountSent == 0)
1811: {
1812:     int z = 0;
1813:     node = xTEDSList.head;
1814:     while(node!=NULL)
1815:     {
1816:         // Bounds check, a device could have registered
1817:         if (z >= SensorMatchesSize)
1818:             break;
1819:
1820:         node->data->inUse->Wait();
1821:         if(SensorMatches[z] > 0 && node->data->getActive() == false && node->data-
>getPosted() == false) //If sensornames is 1 or greater this xTED had matches and xTED is inactive
1822:         {
1823:             debug_f(1,"ReqReg match an inactive task, posting task now... \n");
1824:             node->data->setPosted(true);
1825:             PostTask(node);
1826:         }
1827:         node->data->inUse->Signal();
1828:         z++;
1829:         node = node->next;
1830:     }
1831: }

```

```

1832: delete [] SensorMatches;
1833: //
1834: // Send out the terminating RegInfo message with only the command byte to signal the end of
SDMRegInfo messages
1835: // If this is a subscription ReqReg, fill the requester's info
1836: if (xTEDref != -1)
1837:     id = SenderId;
1838: //
1839: // Send the empty RegInfo message
1840: msgInfo.SendEmpty(id);
1841: MessageSent(&msgInfo);
1842: debug_f(1, "\n");
1843: }
1844:
1845: /*
1846:  Description:
1847:      Handling of the SDMSearch message.  There are two forms of Search calls, one for a
requesting application to
1848:      perform an exhaustive search of the xTEDS, and one for an xTEDS registration change, to
check only interested
1849:      xTEDS subscriptions.  These types of Search calls depend on the xTEDSref parameter (-1 for
an exhaustive
1850:      search, or an index reference number for a registration change.)
1851:
1852:  Input:
1853:      MsgBuf - the message in an array
1854:      size - the length of the message in bytes
1855:      ipaddr - the ip address in dot notation
1856:      xTEDref - an index reference to the xTEDS node
1857:
1858:  Output:
1859:      None
1860:
1861:  Changed:
1862:      None
1863:
1864: */
1865: void Search(char *MsgBuf, int size, const SDMComponent_ID& SenderId, int xTEDref)
1866: {
1867:     xTEDSLibraryListNode* node;
1868:     SDMSearch msgSearch;
1869:     SDMSearchReply msgReply;

```

```

1870:  SDMComponent_ID RecipientId;
1871:  //
1872:  // Unmarshal message
1873:  if(msgSearch.Unmarshal(MsgBuf) < 0)
1874:  {
1875:      printf("Invalid SDMSearch message \n \n");
1876:      return;
1877:  }
1878:  MessageReceived(&msgSearch);
1879:
1880:  //
1881:  // Set the recipient address information
1882:  RecipientId.setPort(msgSearch.destination.getPort());
1883:  if(RecipientId.getPort() == 0)
1884:  {
1885:      printf("Search Error:  Unable to complete SDMSearch because of invalid destination port
\n");
1886:      return;
1887:  }
1888:  if(msgSearch.destination.getAddress() == 0)
1889:      RecipientId.setAddress(SenderId.getAddress());
1890:  else
1891:      RecipientId.setAddress(msgSearch.destination.getAddress());
1892:
1893:  if (debug >= 1)
1894:  {
1895:      char strRequesterAddr[64];
1896:      char strRequesterName[64];
1897:      xTEDSList.GetDeviceName(msgSearch.destination, strRequesterName,
sizeof(strRequesterName));
1898:      msgSearch.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
1899:      debug_f(1, "SDMSearch from %s (%s) reg_expr: \"%s\" \n",
strRequesterAddr, strRequesterName, msgSearch.reg_expr);
1900:
1901:  }
1902:
1903:  //
1904:  // The reply is one of SDM_SEARCH_CURRENT_AND_FUTURE or
SDM_SEARCH_CANCEL for this to match
1905:  if(msgSearch.reply != SDM_SEARCH_CURRENT) //A subscription is wanted or needs to be
canceled
1906:  {
1907:      SDMComponent_ID SubscriberId;

```

```

1908:     SubscriberId.setAddress(SenderId.getAddress());
1909:     SubscriberId.setPort(RecipientId.getPort());
1910:
1911:     pthread_mutex_lock(&subscription_mutex);
1912:     bool AddResult = subscribers.addOrRemoveSubscription(msgSearch.reply, SubscriberId,
msgSearch.source, NULL, NULL, msgSearch.reg_expr, NULL, msgSearch.id, SEARCH_REPLY,
debug);
1913:     pthread_mutex_unlock(&subscription_mutex);
1914:     if(AddResult == false)
1915:     {
1916:         return;
1917:     }
1918: #ifdef PNP_BACKUP
1919:     // Send the Search subscription to all backup DMs
1920:     pthread_mutex_lock(&dm_list_mutex);
1921:     msgSearch.destination = SubscriberId;
1922:     backupDMList.SendMessageToAll(msgSearch);
1923:     pthread_mutex_unlock(&dm_list_mutex);
1924: #endif
1925: }
1926: //
1927: // If this Search call is due to a recent registration, find the node of interest
1928: if(xTEDref != -1)
1929: {
1930:     int xTEDSCount = 0;
1931:     node = xTEDSList.head;
1932:     while(xTEDref != xTEDSCount && node != NULL)
1933:     {
1934:         xTEDSCount++;
1935:         node = node->next;
1936:     }
1937:     if(node == NULL)
1938:     {
1939:         printf("Unable to do Search due to invalid xTEDS reference \n");
1940:         return;
1941:     }
1942: }
1943: else
1944:     node = xTEDSList.head;
1945: //
1946: // Perform the actual search

```

```

1947:  RegexResult SearchResults;
1948:  RegularExpression Pattern;
1949:  if (!Pattern.Set(msgSearch.reg_expr))
1950:  {
1951:      debug_f(0, "Invalid regular expression \"%s\" \n", msgSearch.reg_expr);
1952:      return;
1953:  }
1954:  while(node!=NULL)
1955:  {
1956:      if(node->data->inUse->Wait() == 0)
1957:      {
1958:          if((node->data->getComponentID()==msgSearch.source
msgSearch.source.getSensorID() == 0) && node->data->getActive() == true)
1959:          {
1960:              try
1961:              {
1962:                  SearchResults = node->data->XtedsRegexSearchCapturesOnly(Pattern);
1963:              }
1964:              catch (SDMRegexException e)
1965:              {
1966:                  debug_f(0, "Search Exception: %s \n", e.Message());
1967:              }
1968:              if (SearchResults.Matched())
1969:              {
1970:                  if (debug >= 1)
1971:                  {
1972:                      char strTargetAddr[64];
1973:                      char strDeviceAddr[64];
1974:                      node->data->getComponentID().IDToString(strDeviceAddr,
sizeof(strDeviceAddr));
1975:                      RecipientId.IDToString(strTargetAddr, sizeof(strTargetAddr));
1976:                      debug_f(1, " xTEDS for %s (%s) matched %d times, sending reply to %s. \n",
node->data->getName(), strDeviceAddr, SearchResults.NumMatches(), strTargetAddr);
1977:                  }
1978:                  msgReply.source = node->data->getComponentID();
1979:                  msgReply.id = msgSearch.id;
1980:
1981:                  for (int i = 0; i < SearchResults.NumMatches(); i++)
1982:                  {
1983:                      SearchResults[i].FillMatchText(msgReply.captured_matches,
sizeof(msgReply.captured_matches));
1984:                      msgReply.Send(RecipientId);

```

```

1985:         }
1986:     }
1987: }
1988:     node->data->inUse->Signal();
1989: }
1990:     if(xTEDref != -1)
1991:         break;
1992:     node = node->next;
1993: }
1994: //
1995: // Send out the terminating SDMSearchReply message
1996: msgReply.SendEmpty(RecipientId);
1997: MessageSent(&msgReply);
1998: debug_f(1, "\n");
1999: }
2000:
2001: /*
2002:  Send an SDMAck message with the specified AckStatus.  If UseComponentId is true, send to the
  idDest component ID.
2003:  Otherwise, send to the AppHandle com handle.  If we're sending to idDest, the request was sent
  from device
2004:  directly on the network.  If we're sending to AppHandle, we're responding to the SDMmessage
  wait loop.  If this
2005:  is not the "Active/Leader" DM, i.e., this is a backup DM, don't send an Ack.
2006:
2007:  Input:
2008:      sAckStatus - The status to send, check SDMmessage.h for possible values
2009:      bUseComponentId - Whether to use the idDest to send the message, or AppHandle
2010:      idDest - The component ID of the device to reply to
2011:      AppHandle - The handle object representing the communication information for the waiting
  application class loop
2012:  Output:
2013:      int - Returns the status of the ack message, or 0 if sending an ACK wasn't required
2014:  */
2015: int SendAckMessage(short sAckStatus, bool bUseComponentId, const SDMComponent_ID&
  idDest, const SDMComHandle& AppHandle, bool bForceSend)
2016: {
2017:     SDMAck msgAck;
2018:     int iResult;
2019:
2020: #ifdef PNP_BACKUP
2021:     // If this is not a "force", and we are a backup, don't send

```

```

2022: if (!bForceSend)
2023: {
2024:     pthread_mutex_lock(&dm_list_mutex);
2025:     bool bIsElectedDm = g_bElectedDm;
2026:     pthread_mutex_unlock(&dm_list_mutex);
2027:
2028:     // If we are a backup DM, an Ack has already been sent
2029:     if (!bIsElectedDm)
2030:         return 0;
2031: }
2032: // Otherwise, send
2033: #endif
2034: debug_f(2, "Sending SDMAck message. \n");
2035: msgAck.error = sAckStatus;
2036: if (bUseComponentId)
2037: {
2038:     debug_f(2, "Sending SDMAck message with code: %i to address: 0x%lx port: %i \n",
msgAck.error, idDest.getAddress(), idDest.getPort());
2039:     iResult = msgAck.SendTo(idDest);
2040: }
2041: else
2042:     iResult = msgAck.Send(AppHandle);
2043:
2044: MessageSent(&msgAck);
2045: return iResult;
2046: }
2047: /*
2048: Description:
2049:     Handling of the SDMxTEDS message
2050:
2051: Input:
2052:     arg - a pointer to an xTEDSPParameter
2053:
2054: Output:
2055:     None
2056:
2057: Changed:
2058:     None
2059:
2060: */
2061: void* xTEDS(void* arg)

```



```

2062: {
2063:  char strDeviceName[XTEDS_MAX_ITEM_NAME_SIZE];
2064:  xTEDSLibraryListNode* flag;
2065:  SecTime start, delay;
2066:  SDM_GetTime(start);
2067:  unsigned short usPort = 0;
2068:  int iDeviceType = 0;          // One of ROBOHUB, APPLICATION, or DEVICE
2069:  SDMxTEDS msgXteds;
2070:  xTEDSPParameters* param;
2071:  SDMComponent_ID RequesterId;
2072:  bool bAckUseDeviceAddress = false;
2073:  //
2074:  // Pull out all of the needed information from the xTEDS parameters
2075:  param = (xTEDSPParameters*) arg;
2076:  const char* buf = param->getBuffer();
2077:  RequesterId = param->getSenderAddress();
2078:  const SDMComHandle& AppComHandle = param->GetComHandle();
2079:
2080:  //
2081:  // Unmarshal the message
2082:  if(msgXteds.Unmarshal(buf) < 0)
2083:  {
2084:      printf("Invalid SDMxTEDS message! \n \n");
2085:      SendAckMessage(SDM_INVALID_XTEDS,      bAckUseDeviceAddress,      RequesterId,
AppComHandle);
2086:      delete(param);
2087:      registerNext = true;
2088:      return NULL;
2089:  }
2090:  MessageReceived(&msgXteds);
2091:
2092:  //
2093:  // Save the IP information about the sender
2094:  if(msgXteds.source.getAddress() != 0)
2095:  {
2096:      RequesterId = msgXteds.source;
2097:      bAckUseDeviceAddress = true;
2098:  }
2099:  else if(RequesterId.getAddress() == L_LOCAL_HOST && Address_DM != L_LOCAL_HOST)
2100:  {
2101:      RequesterId.setAddress(Address_DM);

```

```

2102:     msgXteds.source.setAddress(Address_DM);
2103: }
2104: else
2105:     msgXteds.source.setAddress(RequesterId.getAddress());
2106:
2107: if (debug >= 1)
2108: {
2109:     char strTargetAddr[64];
2110:     msgXteds.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
2111:     debug_f(1, "SDMxTEDS from %s length %d PID: %ld \n",
2112:         strTargetAddr, strlen(msgXteds.xTEDS), msgXteds.getPID());
2113:
2114:     if(debug >= 1 && strlen(msgXteds.SPA_node) != 0)
2115:         debug_f(4, " SPA_node: %s \n",msgXteds.SPA_node);
2116: }
2117:
2118: usPort = msgXteds.source.getPort();
2119: //
2120: // Determine if this is a segmented xTEDS
2121: pthread_mutex_lock(&SegmentBuilderMutex);
2122: if (xTEDSBuilder.IsSegmentedxTEDS(msgXteds))
2123: {
2124:     bool xTEDSFinished = false;
2125:     //This xTEDS is segmented, apply this segment
2126:     if (xTEDSBuilder.ApplySegment(msgXteds))
2127:     {
2128:         //Check to see if the xTEDS is finished
2129:         if (xTEDSBuilder.CheckIsFinished(msgXteds))
2130:         {
2131:             //Pull the xTEDS out of the segment builder
2132:             if (xTEDSBuilder.GetFullxTEDS(msgXteds, msgXteds.xTEDS,
2133:                 sizeof(msgXteds.xTEDS)))
2134:                 xTEDSFinished = true;
2135:             //Error occurred
2136:             else
2137:                 printf("Data Manager could not retrieve the segmented xTEDS. \n \n");
2138:         }
2139:         debug_f(1,"xTEDS segment received. \n \n");
2140:         //
2141:         //Send back an SDMAck for this message
2142:         SendAckMessage(SDM_OK, bAckUseDeviceAddress, RequesterId, AppComHandle);

```

```

2142:     }
2143:     //Segment apply failed for some reason
2144:     else
2145:     {
2146:         printf("Error: xTEDS segment could not be stored. \n \n");
2147:         //TODO: Send nack or just timeout?
2148:     }
2149:     //
2150:     //If the xTEDS is not finished, stop here
2151:     if (!xTEDSFinished)
2152:     {
2153:         pthread_mutex_unlock(&SegmentBuilderMutex);
2154:         delete (param);
2155:         registerNext = true;
2156:         return NULL;
2157:     }
2158:     //Otherwise, drop through and continue to parse the xTEDS
2159: }
2160: pthread_mutex_unlock(&SegmentBuilderMutex);
2161: //
2162: //These are a few simple checks to determine if a valid xTEDS document was received correctly,
these operations are cheaper
2163: //than determining the same information during a full parse.
2164: iDeviceType = ParseDeviceName(msgXteds.xTEDS, strDeviceName /*return value*/,
sizeof(strDeviceName), debug); //Get the device name
2165: if(iDeviceType < 0) //If the device or application name was not found
2166: {
2167:     printf("Invalid xTEDS format, xTED will not be registered! Missing Device or Application
name. \n \n"); //This currently happens if the device or application name cannot be found
2168:     SendAckMessage(SDM_INVALID_XTEDS, bAckUseDeviceAddress, RequesterId,
AppComHandle);
2169:     delete(param);
2170:     registerNext = true;
2171:     return NULL;
2172: }
2173: if(msgXteds.xTEDS[0] != '<')
2174: {
2175:     printf("Invalid xTEDS format, xTED will not be registered! Missing opening. \n \n"); //This
currently happens if the xTEDS is not started properly
2176:     SendAckMessage(SDM_INVALID_XTEDS, bAckUseDeviceAddress, RequesterId,
AppComHandle);
2177:     delete(param);

```

```

2178:     registerNext = true;
2179:     return NULL;
2180: }
2181: if(strstr(&msgXteds.xTEDS[0], "</xTEDS>") == NULL)
2182: {
2183:     printf("Invalid xTEDS format, xTED will not be registered! Missing closing </xTEDS> \n
\n"); //This currently happens if the xTEDS is not ended properly
2184:     SendAckMessage(SDM_INVALID_XTEDS, bAckUseDeviceAddress, RequesterId,
AppComHandle);
2185:     delete(param);
2186:     registerNext = true;
2187:     return NULL;
2188: }
2189: addLibrary.Wait();
2190: int iSpot = 0; //Reference index for xTEDSList node
2191: flag = AlreadyRegistered(strDeviceName, buf, usPort, RequesterId, iDeviceType, iSpot /*return
value*/);
2192: if(iSpot == -1)
2193: { // This device is already registered
2194:     SDMID idMessage; //Send a registration confirmation as well as inform component of its
Component_ID
2195:     idMessage.destination = RequesterId;
2196:     idMessage.destination.setPort(usPort);
2197:     idMessage.destination.setSensorID(flag->data->getSensorID());
2198:     idMessage.SendTo(idMessage.destination);
2199:
2200:     addLibrary.Signal();
2201:     SendAckMessage(SDM_OK, bAckUseDeviceAddress, RequesterId, AppComHandle);
2202:     delete(param);
2203:     registerNext = true;
2204:     return NULL;
2205: }
2206:
2207: if(flag == NULL)
2208: {
2209:     if(strncmp(strDeviceName, "RoboHub", 7) == 0 || strncmp(strDeviceName, "RobustHub", 9)
== 0)
2210:     {
2211:         debug_f(3, "Robust Hub being registered \n");
2212:         iDeviceType = ROBOHUB;
2213:     }
2214: #ifdef BUILD_FOR_XTEDS_MERGING

```

```

2215:     bool MergeResult = MergexTEDS(flag, false, msgXteds, iSpot, iDeviceType);
2216: #endif
2217:
2218:     flag = StoreInfo(strDeviceName, iSpot, RequesterId, buf, iDeviceType);
2219:
2220: #ifdef BUILD_FOR_XTEDS_MERGING
2221:     flag->data->setMerged(MergeResult);
2222:     if (MergeResult == true)
2223:         debug_f(1,"Device xTEDS successfully merged with configuration file. \n");
2224: #endif
2225:     debug_f(5,"Finished Storing the info about the xTEDS \n");
2226:
2227:     if(flag->data->setxTEDS(msgXteds.xTEDS) == false)
2228:     {
2229:         bool IsASIM = IsPIDFromASIM(msgXteds.getPID());
2230:         bool IsxTEDSValid = false;
2231:
2232:         // Try DM's saved version of xTEDS for that PID, if it exists
2233:         if (IsASIM)
2234:         {
2235:             //If the xTEDS exists
2236:             if (Retrieve_xTEDS (msgXteds.getPID(), msgXteds.xTEDS))
2237:             {
2238:                 //If the xTEDS parsed correctly
2239:                 if(flag->data->setxTEDS(msgXteds.xTEDS))
2240:                     IsxTEDSValid = true;
2241:             }
2242:             else
2243:                 printf(" \txTEDS: Failed to retrieve stored ASIM xTEDS! \n \n");
2244:         }
2245:
2246:         if (!IsxTEDSValid)
2247:         {
2248:             flag->data->setAvailable(true);
2249:             flag->data->setActive(false);
2250:             flag->data->setMerged(false);
2251:             flag->data->inUse->Signal();
2252:             //xTED was invalid
2253:             printf(" \txTEDS was invalid! \n \n");
2254:             addLibrary.Signal();

```

```

2255:         SendAckMessage(SDM_INVALID_XTEDS, bAckUseDeviceAddress, RequesterId,
AppComHandle);
2256:         delete(param);
2257:         registerNext = true;
2258:         return NULL;                                // Found problem with xTEDS
2259:     }
2260: }
2261: else
2262: {
2263:     //If this is an ASIM, try to store the xTEDS
2264:     if (IsPIDFromASIM(msgXteds.getPID()) && !Store_xTEDS(msgXteds.getPID(),
msgXteds.xTEDS))
2265:     {
2266:         printf(" \txTEDS: Failed to Store xTEDS! \n \n");
2267:     }
2268: }
2269:
2270:     flag->data->inUse->Signal();
2271: }
2272: #ifdef BUILD_FOR_XTEDS_MERGING
2273: //Check for any xTEDS that might need to be merged if this was a robohub that registered
2274: int n = 0;
2275: xTEDSLibraryListNode* node;
2276: if(flag->data->getHub() == ROBOHUB && flag->data->getMerged() == true)
2277: {
2278:     node = xTEDSList.head;
2279:     SDMxTEDS update;
2280:     while(node!=NULL)        // search through the registered items non-merged xTEDS....
2281:     {
2282:         if(node != flag)
2283:             node->data->inUse->Wait();
2284:         if(node->data->getMerged() == false && node->data->getHub() == DEVICE) //Look for
unmerged devices
2285:         {
2286:             //Try to merge due to new registration
2287:             strncpy(update.xTEDS,node->data->getXTEDS(),sizeof(update.xTEDS));
2288:             strncpy(update.SPA_node,node->data->getSPANode(),sizeof(update.SPA_node));
2289:             node->data->inUse->Signal();
2290:             if(MergexTEDS(node,true,update,n,1))
2291:             {
2292:                 debug_f(1,"Device (%s) xTEDS successfully merged with configuration file.
\n",node->data->getName());

```

```

2293:         }
2294:     }
2295:     else
2296:         node->data->inUse->Signal();
2297:     node = node->next;
2298:     n++;
2299: }
2300: }
2301: #endif // #ifdef BUILD_FOR_XTEDS_MERGING
2302: debug_f(4,"Checking for any subscriptions \n");
2303:
2304: flag->data->inUse->Wait();
2305: bool IsActive = flag->data->getActive();
2306: unsigned long ulSID = flag->data->getSensorID();
2307: flag->data->inUse->Signal();
2308: //Get the index to the xTEDS node
2309: int iXtedsRefIndex = (ulSID >> 16) & 0xFF;
2310:
2311: #ifdef PNP_BACKUP
2312:     if(IAmElected())
2313:     {
2314:         //send the xTEDS to all known DM's
2315:         flag->data->inUse->Wait();
2316:         msgXteds.source.setSensorID(flag->data->GetComponentID().getSensorID());
2317:         flag->data->inUse->Signal();
2318:
2319:         pthread_mutex_lock(&dm_list_mutex);
2320:         backupDMList.SendMessageToAll(msgXteds);
2321:         pthread_mutex_unlock(&dm_list_mutex);
2322:     }
2323: #endif
2324:
2325: if(usPort != PORT_DM)
2326: {
2327:     SDMID idMessage; //Send a registration confirmation as well as inform component of its
Component_ID
2328:     idMessage.destination = RequesterId;
2329:     idMessage.destination.setPort(usPort);
2330:     idMessage.destination.setSensorID(ulSID);
2331:     idMessage.SendTo(idMessage.destination);
2332:     char strTargetAddr[64];

```

```

2333:   idMessage.destination.IDToString(strTargetAddr, sizeof(strTargetAddr));
2334:   SDM_GetTime(delay);
2335:   delay -= start;
2336:   debug_f(2, "after %d ms, SDMID sent to %s \n",
delay.GetSeconds()*1000+delay.GetUSconds()/1000, strTargetAddr);
2337: }
2338:
2339:
2340: SendAckMessage(SDM_OK, bAckUseDeviceAddress, RequesterId, AppComHandle);
2341: delete(param);
2342: PrintxTEDS();
2343:
2344: //See if there are any previously interested subscribers for this new registration
2345: if (IsActive && IAmElected())
2346: {
2347:     PublishNotification(ulSID, NOTI_REGISTRATION);
2348:     PublishNotification(ulSID, NOTI_REGISTRATION_CHANGE,
REGISTRATION_MODIFICATION);
2349:     PublishxTEDSModificationSubscription(ulSID, SEARCH_REPLY,
MOD_NOT_APPLICABLE, iXtedsRefIndex);
2350:     PublishxTEDSModificationSubscription(ulSID, REQ_REG_FUTURE,
MOD_NOT_APPLICABLE, iXtedsRefIndex);
2351:     PublishxTEDSModificationSubscription(ulSID, VAR_REQ_REPLY,
MOD_NOT_APPLICABLE, iXtedsRefIndex);
2352: }
2353:
2354: //addLibrary must be held so that there are no state changes while handling subscriptions or
printing xTEDS
2355: addLibrary.Signal();
2356: registerNext = true;
2357: return NULL;
2358: }
2359:
2360: /*
2361: Description:
2362:     Handling of the SDMCancelxTEDS message
2363:
2364: Input:
2365:     arg - a pointer to an xTEDSParameter
2366:
2367: Output:
2368:     None

```



```

2369:
2370: Changed:
2371:     None
2372:
2373: */
2374: void* CancelxTEDS(void* arg)
2375: {
2376:     /* remove xTEDS info from storage */
2377:     unsigned long ulTempID;
2378:     SDMCancelxTEDS msgCancel;
2379:     xTEDSLibraryListNode* node;
2380:     xTEDSParameters* param;
2381:     SDMComponent_ID RequesterId;
2382:     bool bAckUseDeviceAddress = false;
2383:
2384:     param = (xTEDSParameters*) arg;
2385:     const char* buf = param->getBuffer();
2386:     RequesterId = param->getSenderAddress();
2387:     const SDMComHandle& AppComHandle = param->GetComHandle();
2388:
2389:     if(msgCancel.Unmarshal(buf) < 0)
2390:     {
2391:         printf("Invalid SDMCancelxTEDS message! \n \n");
2392:         SendAckMessage(SDM_INVALID_CANCEL,    bAckUseDeviceAddress,    RequesterId,
AppComHandle);
2393:         delete (param);
2394:         cancelNext = true;
2395:         return 0;
2396:     }
2397:     MessageReceived(&msgCancel);
2398:
2399:     // Copy sensor ID and port into a temp for string comparison
2400:     unsigned long ulSID = msgCancel.source.getSensorID();
2401:     unsigned short usPort = msgCancel.source.getPort();
2402:     //
2403:     // Set the address information
2404:     if(msgCancel.source.getAddress() != 0)
2405:     {
2406:         RequesterId.setAddress(msgCancel.source.getAddress());
2407:         //TODO:
2408:         // Applications will usually fill in msgCancel.source.getAddress, how

```

```

2409:    // to distinguish between devices and apps?
2410:    //bAckUseDeviceAddress = true;
2411: }
2412: else if(RequesterId.getAddress() == L_LOCAL_HOST && Address_DM != L_LOCAL_HOST)
2413: {
2414:     RequesterId.setAddress(Address_DM);
2415:     msgCancel.source.setAddress(Address_DM);
2416: }
2417: else
2418:     msgCancel.source.setAddress(RequesterId.getAddress());
2419:
2420: if (debug >= 1)
2421: {
2422:     char strTargetAddr[64];
2423:     msgCancel.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
2424:     debug_f(1, "SDMCancelxTEDS for %s \n", strTargetAddr);
2425: }
2426:
2427: //
2428: // Start looking for the xTEDS to cancel
2429: node = xTEDSList.head;
2430: int iRemoveRef = 0;
2431: while(node!=NULL)
2432: {
2433:     node->data->inUse->Wait();
2434:     if(node->data->getAvailable() == false)
2435:     {
2436:         // PORT_PM only matches if a PID was also present, otherwise the PM may accidentally
2437:         // cancel a persistent application's xTEDS whose SID was sent as zero, when
2438:         // the PM cancels itself
2439:         // Also, if the port and sensor id are zero, this is the Network Manager cancelling a
2440:         // SPA-S ASIM's xTEDS when it detects a disconnect
2441:         if(usPort == node->data->getTargetPort() || (usPort == PORT_PM && ulSID != 0) ||
(usPort == 0 && ulSID == 0))
2442:         {
2443:             if (RequesterId.getAddress() == node->data->getAddress())
2444:             {
2445:                 if(node->data->getActive() == true)
2446:                 {
2447:                     ulTempID = node->data->getSensorID();    //Get sensor ID DM created

```

```

2448:                ulTempID &= 0x000000FF;//Change DM sensor ID to ID assigned by SM
or application
2449:                // Cancel the xTEDS corresponding to the user-specified SID
2450:                if((ulSID&0x000000FF) == ulTempID || (usPort == PORT_PM && node-
>data->getTargetPort() != PORT_DM && node->data->getTargetPort() != PORT_SM && node->data-
>getTargetPort() != PORT_TM && node->data->getTargetPort() != PORT_SPA1_MANAGER))
2451:                {
2452:                    if(usPort == PORT_PM)
2453:                    {
2454:                        // If the PM is cancelling an xTEDS
2455:                        if(ulSID == node->data->getPid())
2456:                        {
2457:                            debug_f(3,"Sensor name to cancel is %s \n", node->data-
>getName());
2458:                            node->data->inUse->Signal();
2459:                            break;
2460:                        }
2461:                    }
2462:                    else
2463:                    {
2464:                        debug_f(3,"Sensor name to cancel is %s \n", node->data-
>getName());
2465:                        node->data->inUse->Signal();
2466:                        break;
2467:                    }
2468:                }
2469:            }
2470:        }
2471:    }
2472: }
2473: iRemoveRef++;
2474: node->data->inUse->Signal();
2475: node = node->next;
2476: }
2477: addLibrary.Wait();
2478: //
2479: // If the xTEDS couldn't be found, report error and exit
2480: if(node==NULL)
2481: {
2482:     debug_f(0,"Could not identify the sensor to cancel. \n");
2483:     SendAckMessage(SDM_UNKNOWN_XTEDS,    bAckUseDeviceAddress,    RequesterId,
AppComHandle);

```

```

2484:     delete(param);
2485:     PrintxTEDS();
2486:     addLibrary.Signal();
2487:     cancelNext = true;
2488:     return 0;
2489: }
2490:
2491: node->data->inUse->Wait();
2492: const unsigned long CancelSensorID = node->data->getSensorID();
2493: const SDMComponent_ID CancelComponentId = node->data->getComponentID();
2494: msgCancel.source.setSensorID(CancelSensorID);
2495: if(!(node->data->getTargetPort() == PORT_SM || node->data->getTargetPort() ==
PORT_SPA1_MANAGER || node->data->getTargetPort() == PORT_TM || node->data->getTargetPort()
== PORT_PM))
2496: {
2497:     node->data->setActive(false);
2498:     node->data->setPosted(false);
2499: }
2500:
2501: if(msgCancel.fullCancel == 1)
2502: {
2503:     node->data->setActive(true);
2504: }
2505: node->data->inUse->Signal();
2506: //
2507: // Publish any subscriptions about the cancellation
2508: #ifdef PNP_BACKUP
2509: if (IAmElected())
2510: {
2511: #endif
2512:     debug_f(4,"Checking for any subscriptions \n");
2513:     PublishxTEDSModificationSubscription(CancelSensorID, REQ_REG_CANCELLATION,
DEREGISTRATION_MODIFICATION, iRemoveRef);
2514:     PublishNotification(CancelSensorID, NOTI_DEREGISTRATION);
2515:     PublishNotification(CancelSensorID, NOTI_REGISTRATION_CHANGE,
DEREGISTRATION_MODIFICATION);
2516: #ifdef PNP_BACKUP
2517: }
2518: #endif
2519: //
2520: // Notify any devices about the cancelled application (ala SDMDeletesub)
2521: pthread_mutex_lock(&sensor_subs_mutex);

```

```

2522: g_SensorSubscriptions.SubscriberFinish(CancelComponentId);
2523: g_SensorSubscriptions.ProviderFinish(CancelComponentId);
2524: pthread_mutex_unlock(&sensor_subs_mutex);
2525:
2526: node->data->inUse->Wait();
2527: if(node->data->getTargetPort()==PORT_DM)
2528: {
2529:     printf("Only the Data Manager can cancel its xTEDS!! \n \n");
2530:     node->data->inUse->Signal();
2531:     SendAckMessage(SDM_INVALID_CANCEL,    bAckUseDeviceAddress,    RequesterId,
AppComHandle);
2532:     delete (param);
2533:     addLibrary.Signal();
2534:     cancelNext = true;
2535:     return 0;
2536: }
2537: if(node->data->getActive() == true)
2538: {
2539:     node->data->setAvailable(true);
2540:     node->data->setTargetPort(0);
2541:     node->data->setConnections(0);
2542:     node->data->setPid(0);
2543:     node->data->setHub(0);
2544:     node->data->setMerged(false);
2545:     node->data->setActive(false);
2546: }
2547: node->data->inUse->Signal();
2548:
2549: #ifdef PNP_BACKUP
2550: //send CancelxTEDS to all known backup DM's
2551: pthread_mutex_lock(&dm_list_mutex);
2552: backupDMList.SendMessageToAll(msgCancel);
2553: pthread_mutex_unlock(&dm_list_mutex);
2554: #endif
2555:
2556: SendAckMessage(SDM_OK, bAckUseDeviceAddress, RequesterId, AppComHandle);
2557: delete(param);
2558: PrintxTEDS();
2559: addLibrary.Signal();
2560: cancelNext = true;
2561: return 0;

```

```

2562: }
2563:
2564: /*
2565:  Cancel a task's xTEDS but don't mark it as unposted. This is because a PM will restart
2566:  the task, and any subscribers should be notified via the DM services, but the TM still
2567:  has it scheduled to the PM node. This prevents the task from being reposted if an
2568:  extern application does SDMReqReg and it matches the unposted task.
2569: */
2570: void TaskError(char* buf, int size, const SDMComponent_ID& RequesterId)
2571: {
2572:     SDMTaskError msgError;
2573:     if (msgError.Unmarshal(buf) < 0)
2574:     {
2575:         printf("Invalid SDMTaskError message. \n");
2576:         return;
2577:     }
2578:     MessageReceived(&msgError);
2579:
2580: #ifdef PNP_BACKUP
2581:     // Send the TaskError message to all backup DMs
2582:     SDMComponent_ID temp = RequesterId;
2583:     msgError.source = RequesterId;
2584:     pthread_mutex_lock(&dm_list_mutex);
2585:     backupDMList.SendMessageToAll(msgError);
2586:     pthread_mutex_unlock(&dm_list_mutex);
2587:     msgError.source = temp;
2588: #endif
2589:
2590:     debug_f(1, "SDMTaskError for \"%s\" pid: %u",
2591:     msgError.filename, msgError.pid);
2592:
2593:     //
2594:     // Start looking for the xTEDS to cancel
2595:     xTEDSLibraryListNode* node = xTEDSList.head;
2596:     int iRemoveRef = 0;
2597:     while(node!=NULL)
2598:     {
2599:         node->data->inUse->Wait();
2600:         if(node->data->getAvailable() == false)
2601:         {
2602:             if (RequesterId.getAddress() == node->data->getAddress())

```

```

2603:     {
2604:         if(node->data->getActive() == true)
2605:         {
2606:             if (node->data->getPid() == msgError.pid)
2607:             {
2608:                 debug_f(3,"Sensor name to cancel is %s \n", node->data->getName());
2609:                 node->data->inUse->Signal();
2610:                 break;
2611:             }
2612:         }
2613:     }
2614: }
2615: iRemoveRef++;
2616: node->data->inUse->Signal();
2617: node = node->next;
2618: }
2619: addLibrary.Wait();
2620: //
2621: // If the xTEDS couldn't be found, report error and exit
2622: if(node==NULL)
2623: {
2624:     debug_f(0,"Could not identify the sensor to cancel. \n");
2625:     addLibrary.Signal();
2626:     return ;
2627: }
2628:
2629: node->data->inUse->Wait();
2630: node->data->setActive(false);
2631: // Keep this task posted so a ReqReg won't re-post it -- the PM will
2632: // immediately restart it
2633: node->data->setPosted(true);
2634: const unsigned long CancelSensorID = node->data->getSensorID();
2635: const SDMComponent_ID& CancelComponentId = node->data->getComponentID();
2636: node->data->inUse->Signal();
2637:
2638: //
2639: // Notify any devices about the cancelled application (ala SDMDeltesub)
2640: pthread_mutex_lock(&sensor_subs_mutex);
2641: g_SensorSubscriptions.SubscriberFinish(CancelComponentId);
2642: g_SensorSubscriptions.ProviderFinish(CancelComponentId);
2643: pthread_mutex_unlock(&sensor_subs_mutex);

```

```

2644:
2645: //Remove any subscriptions that the failed task had
2646: pthread_mutex_lock(&subscription_mutex);
2647: subscribers.removeReqRegSubscription(11, CancelComponentId, debug);
2648: pthread_mutex_unlock(&subscription_mutex);
2649:
2650: PrintxTEDS();
2651: //
2652: // Publish any subscriptions about the cancellation
2653: if (IAmElected())
2654: {
2655:     debug_f(4,"Checking for any subscriptions \n");
2656:     PublishxTEDSModificationSubscription(CancelSensorID, REQ_REG_CANCELLATION,
DEREGISTRATION_MODIFICATION, iRemoveRef);
2657:     PublishNotification(CancelSensorID, NOTI_DEREGISTRATION);
2658:     PublishNotification(CancelSensorID, NOTI_REGISTRATION_CHANGE,
DEREGISTRATION_MODIFICATION);
2659: }
2660:
2661:
2662: addLibrary.Signal();
2663: return ;
2664: }
2665:
2666: /*
2667: Description:
2668:     Handling of the SDMReqxTEDS message
2669:
2670: Input:
2671:     buf - the message in an array
2672:     size - the length of the message in bytes
2673:     ipaddr - the ip address in dot notation
2674:     dport - the port the message came from
2675:
2676: Output:
2677:     None
2678:
2679: Changed:
2680:     None
2681:
2682: */

```



```

2683: void ReqxTEDS(char *buf, int size, const SDMComponent_ID& SenderId)
2684: {
2685:     int iSelect = 0, found = 0, length = 0;
2686:     unsigned long sID = 0;
2687:     char msg[LARGE_MSG_BUFSIZE];
2688:     unsigned short usPort = 0;
2689:     int reply = 0;
2690:     int temp = 0;
2691:     SDMReqxTEDS msgRequest;
2692:     SDMxTEDSInfo msgInfo;
2693:     xTEDSLibraryListNode* node;
2694:     SDMComponent_ID id;
2695:
2696:     if(msgRequest.Unmarshal(buf) < 0)
2697:     {
2698:         printf("Invalid SDMReqxTEDS message! \n \n");
2699:         return;
2700:     }
2701:     MessageReceived(&msgRequest);
2702:
2703:     memset(msg, 0, LARGE_MSG_BUFSIZE);
2704:     //Get ip address where message sent from
2705:     iSelect = msgRequest.select;
2706:     iSelect &= 0x01;
2707:     reply = msgRequest.select;
2708:     reply &= 0x02;
2709:     usPort = msgRequest.destination.getPort();
2710:     if(reply == 0)
2711:         usPort = SenderId.getPort();
2712:     if(msgRequest.destination.getAddress() == 0)
2713:         id.setAddress(SenderId.getAddress());
2714:     else
2715:         id.setAddress(msgRequest.destination.getAddress());
2716:     id.setPort(usPort);
2717:     temp = msgRequest.select;
2718:
2719:     if(temp > 3)
2720:     {
2721:         printf("Select byte is not set to a valid value, cannot complete request! \n \n");
2722:         return;
2723:     }

```

```

2724: if(iSelect == 0)    //select byte is set to use sensorID
2725: {
2726:     sID = msgRequest.source.getSensorID();
2727:     if (debug >= 1)
2728:     {
2729:         char strRequesterAddr[64];
2730:         char strTargetAddr[64];
2731:         msgRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
2732:         msgRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
2733:         debug_f(1, "SDMReqxTEDS for %s from %s select: %d \n",
2734:             strTargetAddr, strRequesterAddr, msgRequest.select);
2735:     }
2736:
2737:     node = MatchSID(sID);
2738:     if(node==NULL)
2739:     {
2740:         printf("No matching sensorID found \n \n");
2741:     }
2742:     else
2743:         found = 1;
2744:
2745:     if(found == 1)
2746:     {
2747:         msgInfo.source.setSensorID(node->data->getSensorID());
2748:         msgInfo.source.setPort(node->data->getTargetPort());
2749:         msgInfo.source.setAddress(node->data->getAddress());
2750:         length = strlen(node->data->getxTEDS()); //Set the length of the xTED
2751:         memcpy(&msgInfo.xTEDS,node->data->getxTEDS(),length);
2752:         msgInfo.xTEDS[length] = '\0'; //Null terminate the msg
2753:         debug_f(1,"Sending xTEDS for %s \n",node->data->getName());
2754:         node->data->inUse->Signal();
2755:         msgInfo.Send(id);
2756:         MessageSent(&msgInfo);
2757:         msgInfo.Marshal(msg);
2758:         msgInfo.SendEmpty(id);
2759:         MessageSent(&msgInfo);
2760:         debug_f(1,"Message sent to app: %c on port %hu \n",msg[0],usPort);
2761:     }
2762:     else
2763:     {
2764:         msgInfo.SendEmpty(id);

```

```

2765:         MessageSent(&msgInfo);
2766:         debug_f(1,"Empty message sent to app: %c \n",msg[0]);
2767:     }
2768: }
2769: else//If xTED found with device_name
2770: {
2771:     if (debug >= 1)
2772:     {
2773:         char strRequesterAddr[64];
2774:         char strTargetAddr[64];
2775:         msgRequest.source.IDToString(strTargetAddr, sizeof(strTargetAddr));
2776:         msgRequest.destination.IDToString(strRequesterAddr, sizeof(strRequesterAddr));
2777:         debug_f(1, "SDMReqxTEDS for %s from %s select: %d device: %s \n",
2778:             strTargetAddr, strRequesterAddr, msgRequest.select,
2779:             msgRequest.device_name);
2780:     }
2781:     node = xTEDSList.head;
2782:     while(node!=NULL)    //Look at each xTED device name
2783:     {
2784:         node->data->inUse->Wait();
2785:         if(node->data->getAvailable() == false) //Check to see if there is an xTED at this array
location
2786:         {
2787:             if(strncmp(msgRequest.device_name,node->data->getName(),strlen(node->data-
>getName()))==0) //Check to see if device name matches
2788:             {
2789:                 found = 1; //Set found to 1
2790:             }
2791:         }
2792:
2793:         if(found == 1) //If a device name matched
2794:         {
2795:             msgInfo.source.setSensorID(node->data->getSensorID());
2796:             msgInfo.source.setPort(node->data->getTargetPort());
2797:             msgInfo.source.setAddress(node->data->getAddress());
2798:             length = strlen(node->data->getxTEDS()); //Set the length of the xTED
2799:             memcpy(&msgInfo.xTEDS,node->data->getxTEDS(),length);
2800:             msgInfo.xTEDS[length] = '\0'; //Null terminate the msg
2801:             debug_f(1,"Sending xTED for %s \n",node->data->getName());
2802:             msgInfo.Marshal(msg);
2803:             msgInfo.Send(id);

```

```

2804:         MessageSent(&msgInfo);
2805:         debug_f(1,"Message sent to app: %c on port %hu \n", msg[0], usPort);
2806:         found = 0; //Set found to 0
2807:     }
2808:     node->data->inUse->Signal();
2809:     node = node->next;
2810: }
2811: msgInfo.SendEmpty(id);
2812: MessageSent(&msgInfo);
2813: debug_f(1,"Message sent to app: %c on port %hu \n", msg[0], usPort);
2814: }
2815: }
2816:
2817: /*
2818:  Description:
2819:      Handling of Subscriptions to various item the DM produces. See the DM xTEDS.
2820:
2821:  Input:
2822:      ipaddress - the ip address of the subscriber in dot form
2823:      port - the port of the subscriber
2824:      mID - the message ID of the subscription to be published
2825:
2826:  Output:
2827:      None
2828:
2829:  Changed:
2830:      None
2831:
2832: */
2833: void Subscribe(const SDMComponent_ID& SubscriberId, const SDMMessage_ID& mID)
2834: {
2835:     xTEDSLibraryListNode* node;
2836:     long ulSID = 0;
2837:     SDMData msgData;
2838:     SubscriptionListNode* result;
2839:
2840:     // Add this subscription
2841:     pthread_mutex_lock(&subscription_mutex);
2842:     result = subscribers.addSubscription(SubscriberId, mID.getInterfaceMessagePair(), debug);
2843:     pthread_mutex_unlock(&subscription_mutex);
2844:     if(result==NULL)

```

```

2845:     return;
2846:
2847: if(!IAmElected())
2848: {
2849:     return;
2850: }
2851:
2852: // For subscriptions to registration events, send all currently registered devices
2853: if(mID == NOTI_REGISTRATION || mID == NOTI_REGISTRATION_CHANGE)
2854: {
2855:     node = xTEDSList.head;
2856:     while(node!=NULL)
2857:     {
2858:         node->data->inUse->Wait();
2859:         if(node->data->getAvailable() == false)
2860:         {
2861:             msgData.source = DataManager;
2862:             msgData.msg_id = mID;    //Copy the mID into msg
2863:
2864:             ulSID = node->data->getSensorID();
2865:             if(mID == NOTI_REGISTRATION)
2866:             {
2867:                 PUT_ULONG(&msgData.msg[0], ulSID); //Copy sID into msg
2868:                 msgData.Send(SubscriberId, 4);
2869:                 MessageSent(&msgData);
2870:
2871:                 debug_f(1,"Sent msg to %s at port %hu sensorID: %lu msgID: 0x%x msg data of
2872: %x %x %x %x \n",result->data->getAddress(),result->data->getPort(),msgData.source.getSensorID(),
2873: msgData.msg_id.getInterfaceMessagePair(), msgData.msg[0],
2874: msgData.msg[1],msgData.msg[2],msgData.msg[3]);
2875:             }
2876:             else
2877:             {
2878:                 unsigned char temp = REGISTRATION_MODIFICATION;
2879:                 PUT_UCHAR(&msgData.msg[0], temp);
2880:                 PUT_ULONG(&msgData.msg[1], ulSID); //Copy sID into msg
2881:                 msgData.Send(SubscriberId, 5);
2882:                 MessageSent(&msgData);
2883:
2884:                 debug_f(1,"Sent msg to %s at port %hu sensorID: %lu msgID: 0x%x msg data of
2885: %x %x %x %x %x \n", result->data->getAddress(), result->data->getPort(),

```

```

msgData.source.getSensorID(),      msgData.msg_id.getInterfaceMessagePair(),      msgData.msg[0],
msgData.msg[1], msgData.msg[2], msgData.msg[3], msgData.msg[4]);
2882:         }
2883:     }
2884:     node->data->inUse->Signal();
2885:     node = node->next;
2886: }
2887: }
2888: else if(mID == NOTI_PERFORMANCE_COUNTERS)
2889: {
2890:     debug_f(0, "Subscription to performance counters at 0x%lx:%hu \n",
SubscriberId.getAddress(), SubscriberId.getPort());
2891:
2892: #ifndef __VXWORKS__
2893:     itimerval interval;
2894:     getitimer(ITIMER_REAL, &interval);
2895:     if (interval.it_value.tv_sec == 0 && interval.it_interval.tv_sec == 0)
2896:     {
2897:         //Time interval for the publish interval of the performance counter
2898:         timeval pubInterval;
2899:         pubInterval.tv_sec = 1;
2900:         pubInterval.tv_usec = 0;
2901:
2902:         itimerval timerInterval;
2903:         timerInterval.it_interval = pubInterval;
2904:         timerInterval.it_value = pubInterval;
2905:
2906:         //Set the performance counter timer
2907:         setitimer (ITIMER_REAL, &timerInterval, NULL);
2908:     }
2909: #else //VxWorks has a different timer API
2910:     itimerspec interval;
2911:     timer_gettime(CLOCK_REALTIME, &interval);
2912:     if (interval.it_value.tv_sec == 0 && interval.it_interval.tv_sec == 0)
2913:     {
2914:         //Time interval for the publish interval of the performance counter
2915:         itimerspec timerInterval;
2916:         timerInterval.it_value.tv_sec = 1;
2917:         timerInterval.it_value.tv_nsec = 0;
2918:
2919:         //Set the performance counter timer

```

```

2920:         timer_settime(CLOCK_REALTIME, 0, &timerInterval, NULL);
2921:     }
2922: #endif
2923: }
2924: return; //Function is finished
2925: }
2926:
2927: /*
2928:  Description:
2929:      Handling of canceling of a subscription that the DM publishes. See the DM xTEDS.
2930:
2931:  Input:
2932:      ipaddress - the ip address of the subscriber
2933:      port - the port of the subscriber
2934:      mID - the message ID of the subscription
2935:
2936:  Output:
2937:      None
2938:
2939:  Changed:
2940:      None
2941:
2942: */
2943: void CancelSubscription(const SDMComponent_ID& SubscriberId, int mID)
2944: {
2945:     SubscriptionListNode* node;
2946:
2947:     pthread_mutex_lock(&subscription_mutex);
2948:     node = subscribers.head;
2949:     while(node!=NULL) //look through subscriptions
2950:     {
2951:         if(node->data->getInuse() == true) //Check to see if in use
2952:         {
2953:             if(node->data->getDestination() == SubscriberId)
2954:             {
2955:                 if(node->data->getmID() == mID) //Check to see if message id matches
2956:                 {
2957:                     if(node->data->getItems() == false)
2958:                     {
2959:                         node->data->setPort(0); //Set port to 0
2960:                         node->data->setmID(0); //Set message id to 0

```

```

2961:         node->data->setInuse(false);    //Set in use to false
2962:         debug_f(2,"Canceled subscription \n \n");
2963:         pthread_mutex_unlock(&subscription_mutex);
2964:         return;
2965:     }
2966: }
2967: else
2968: {
2969:     debug_f(4,"Unable to cancel subscription because message ID does not match
current subscriptions \n");
2970: }
2971: }
2972: else
2973: {
2974:     debug_f(4,"Unable to cancel subscription because address does not match current
subscription \n");
2975: }
2976: }
2977:     node = node->next;
2978: }
2979: pthread_mutex_unlock(&subscription_mutex);
2980: printf("The DM was unable to cancel your subscription because there was no subscription
matching your Cancel request \n");
2981: }
2982:
2983: /*
2984:  Description:
2985:      Handling of publishing messages that the DM generates. This function handles the DM's
subscriptions in its xTEDS as well
2986:      as subscriptions that are generated from the FUTURE ReqReg and Search queries. For the
latter subscriptions, this function
2987:      depends on calling again the ReqReg and Search functions in order for the publication, but
sets the xTEDref variable so it
2988:      doesn't perform an exhaustive search.
2989:
2990:  Input:
2991:      node - a pointer to a subscription list node
2992:      sID - generally the sensorID of the device in question
2993:      mID - the message ID of the subscription
2994:      xTEDref - an index to the xTEDS node
2995:
2996:  Output:

```



```

2997:     None
2998:
2999: Changed:
3000:     None
3001:
3002: */
3003:
3004: void PublishxTEDSModificationSubscription(unsigned long SensorID, int MessageID, int
ModificationAction, int xTEDSRef)
3005: {
3006:     char MessageBuffer[BUFSIZE];
3007:     int MessageSize = 0;
3008:     SDMComponent_ID SubscriberID;
3009:     SubscriptionListNode* CurSub = NULL;
3010:
3011:     pthread_mutex_lock(&subscription_mutex);
3012:     CurSub = subscribers.head;
3013:     int n = 0;
3014:     while(CurSub!=NULL) //Check for matching registration subscriptions
3015:     {
3016:         // If the current subscription node is in use and the message id matches
3017:         if(CurSub->data->getInuse() == true && CurSub->data->getmID() == MessageID)
3018:         {
3019:             debug_f(4,"subscribers[%d].submID 0x%x\n", n, CurSub->data->getmID());
3020:             debug_f(4,"Publishing message ID of 0x%x\n",MessageID);
3021:
3022:             SubscriberID = CurSub->data->getDestination();
3023:
3024:             if(MessageID == REQ_REG_FUTURE || MessageID ==
REQ_REG_CANCELLATION)//If mID matches
3025:             {
3026:                 //If this is a REQREG, or REQREG_CANCEL
3027:                 if(CurSub->data->getItems() == true) //If subscription has items
3028:                 {
3029:                     debug_f(4, "Performing subscriber ReqReg with action %d
\n",ModificationAction);
3030:                     // Put together a SDMReqReg message
3031:                     SDMReqReg req;
3032:                     req.reply = SDM_REQREG_CURRENT;
3033:                     req.destination = SubscriberID;
3034:                     req.id = CurSub->data->getID();
3035:                     req.source = CurSub->data->getSource();

```

```

3036:
3037:         // Copy item_name into msg
3038:         memcpy(req.item_name, CurSub->data->getItemName(), strlen(CurSub->data-
>getItemName()+1);
3039:
3040:         // Copy qual_list into msg
3041:         memcpy(req.qual_list, CurSub->data->getQuallist(), strlen(CurSub->data-
>getQuallist()+1);
3042:
3043:         // Copy device into msg
3044:         memcpy(req.device, CurSub->data->getDevice(), strlen(CurSub->data-
>getDevice()+1);
3045:
3046:         // Copy interface into msg
3047:         memcpy(req.interface, CurSub->data->getInterface(), strlen(CurSub->data-
>getInterface()+1);
3048:
3049:         MessageSize = req.Marshal(MessageBuffer);
3050:
3051:         pthread_mutex_unlock(&subscription_mutex);
3052:         ReqReg(MessageBuffer,      MessageSize,      xTEDSRef,      SubscriberID,
ModificationAction);
3053:         pthread_mutex_lock(&subscription_mutex);
3054:     }
3055: }
3056: else if(MessageID == SEARCH_REPLY) //If submID is for a Search
3057: {
3058:     debug_f(4, "Performing subscriber Search \n");
3059:     SDMSearch sear;
3060:     sear.reply = SDM_SEARCH_CURRENT;
3061:     sear.destination.setPort(CurSub->data->getPort());
3062:     sear.id = CurSub->data->getID();
3063:     sear.source = CurSub->data->getSource();
3064:
3065:     memcpy(sear.reg_expr,      CurSub->data->getItemName(),      strlen(CurSub->data-
>getItemName()+1);
3066:     MessageSize = sear.Marshal(MessageBuffer);
3067:
3068:     pthread_mutex_unlock(&subscription_mutex);
3069:     Search(MessageBuffer, MessageSize, SubscriberID, xTEDSRef);
3070:     pthread_mutex_lock(&subscription_mutex);
3071: }

```

```

3072:         else if(MessageID == VAR_REQ_REPLY)
3073:         {
3074:             debug_f(4,"Performing subscriber VarInfo \n");
3075:             SDMVarReq var;
3076:             var.reply = SDM_VARREQ_CURRENT;
3077:             var.destination.setPort(CurSub->data->getPort());
3078:             var.id = CurSub->data->getID();
3079:             var.source = CurSub->data->getSource();
3080:             var.msg_id = atoi(CurSub->data->getInterface());
3081:
3082:             memcpy(var.variable,    CurSub->data->getItemName(),    strlen(CurSub->data-
>getItemName()+1);
3083:             MessageSize = var.Marshal(MessageBuffer);
3084:
3085:             pthread_mutex_unlock(&subscription_mutex);
3086:             VarReq(MessageBuffer, MessageSize, SubscriberID, xTEDSRef);
3087:             pthread_mutex_lock(&subscription_mutex);
3088:         }
3089:     }
3090:     CurSub = CurSub->next;
3091:     n++;
3092: }
3093: pthread_mutex_unlock(&subscription_mutex);
3094: }
3095: void PublishNotification(unsigned long SensorID, SDMMMessage_ID MessageID, int
ModificationAction)
3096: {
3097:     SDMData msgData;
3098:     msgData.msg_id = MessageID;
3099:     msgData.source = DataManager;
3100:
3101:     debug_f(3, "Attempting to publish a notification for interface %hhd message %hhd. \n",
MessageID.getInterface(), MessageID.getMessage());
3102:
3103:     pthread_mutex_lock(&subscription_mutex);
3104:     SubscriptionListNode* CurSub = subscribers.head;
3105:     while (CurSub != NULL)
3106:     {
3107:         if (CurSub->data->getInuse() == true && CurSub->data->getmID() ==
MessageID.getInterfaceMessagePair())
3108:         {

```

```

3109:         if (MessageID == NOTI_REGISTRATION || MessageID ==
NOTI_DEREGISTRATION || MessageID == NOTI_XTEDS_MODIFICATION)
3110:         {
3111:             debug_f(3, " Publishing a registration, deregistration, or modification notification.
\n");
3112:             PUT_ULONG (msgData.msg, SensorID);
3113:             msgData.Send(CurSub->data->getDestination(), sizeof(unsigned long));
3114:             MessageSent(&msgData);
3115:         }
3116:     else if (MessageID == NOTI_REGISTRATION_CHANGE)
3117:     {
3118:         debug_f(3, " Published a registration change notification. \n");
3119:         unsigned char temp = (unsigned char)ModificationAction;
3120:         PUT_UCHAR (msgData.msg, temp);
3121:         PUT_ULONG (msgData.msg+1, SensorID);
3122:         msgData.Send(CurSub->data->getDestination(), sizeof(unsigned long) + 1);
3123:         MessageSent(&msgData);
3124:     }
3125:     else if (MessageID == NOTI_PERFORMANCE_COUNTERS)
3126:     {
3127:         debug_f(3, " Publishing performance counters. \n");
3128:         pthread_mutex_lock(&perf_counter_mutex);
3129:         PUT_UINT(&msgData.msg[0], total_recd);
3130:         PUT_UINT(&msgData.msg[4], prevsec_recd);
3131:         PUT_UINT(&msgData.msg[8], total_sent);
3132:         PUT_UINT(&msgData.msg[12], prevsec_sent);
3133:         pthread_mutex_unlock(&perf_counter_mutex);
3134:         msgData.Send(CurSub->data->getDestination(),16);
3135:         MessageSent(&msgData);
3136:     }
3137: }
3138: CurSub = CurSub->next;
3139: }
3140: pthread_mutex_unlock(&subscription_mutex);
3141: debug_f(4, "Finished looking for notification subscriptions. \n");
3142: }
3143:
3144: /*
3145: Description:
3146: Handling of the SDMReady message
3147:

```

```

3148: Input:
3149:     buf - the message in an array
3150:     size - the length of the message in bytes
3151:     ipaddr - the ip address in dot notation
3152:
3153: Output:
3154:     None
3155:
3156: Changed:
3157:     None
3158:
3159: */
3160: void Ready(char *buf, int size, const SDMComponent_ID& SenderId) //Send a reply that the DM
is ready
3161: {
3162:     unsigned short usPort = 0;
3163:     SDMReady msgReady;
3164:     SDMComponent_ID id;
3165:
3166:     //Get ip address where message sent from
3167:     if(msgReady.Unmarshal(buf) <0)
3168:     {
3169:         printf("Invalid SDMReady message! \n \n");
3170:         return;
3171:     }
3172:     MessageReceived(&msgReady);
3173:
3174:     usPort = msgReady.destination.getPort(); //Save the port to communicate on
3175:
3176:     if (debug >= 1)
3177:     {
3178:         char strRequesterId[64];
3179:         msgReady.destination.IDToString(strRequesterId, sizeof(strRequesterId));
3180:         debug_f(1, "SDMReady from %s \n", strRequesterId);
3181:     }
3182:
3183:     if(usPort == PORT_TM)
3184:     {
3185:         TaskManager = msgReady.destination;
3186:         debug_f(1, "Setting TM ip address to 0x%lx \n", TaskManager.getAddress());
3187:

```

```

3188:
3189: #ifdef PNP_BACKUP
3190:     xTEDSLibraryListNode* node = xTEDSList.head;
3191:     while(node != NULL)
3192:     {
3193:         node->data->inUse->Wait();
3194:         if (strcmp(node->data->getName(), "TaskManager", 11) == 0)
3195:         {
3196:             printf("Updating the registered TM Address to 0x%lx \n", TaskManager.getAddress());
3197:             node->data->setAddress(TaskManager);
3198:         }
3199:         node->data->inUse->Signal();
3200:         node = node->next;
3201:     }
3202:
3203:     backupDMLList.SendMessageToAll(msgReady);
3204:     printf("Forwarding the TM address to backups \n");
3205:     //backupDMLList.PrintList();
3206: #endif
3207: }
3208: id.setAddress(msgReady.destination.getAddress());
3209: id.setPort(usPort);
3210: msgReady.destination.setAddress(DataManager.getAddress());
3211: msgReady.destination.setPort(PORT_DM);
3212: if(usPort == PORT_DM_ELECTION)
3213: {
3214:     if(TaskManager.getPort() != 0)
3215:     {
3216:         msgReady.source = TaskManager;
3217:     }
3218:     //msgReady.Send(id);
3219: #ifdef PNP_BACKUP
3220:     SDMComponent_ID BackupId = id;
3221:     BackupId.setPort(PORT_DM_ELECTION);
3222:
3223:     //Check if new backup DM and store the ip
3224:     pthread_mutex_lock(&dm_list_mutex);
3225:     bool NewAdd = backupDMLList.AddIfNew(BackupId);
3226:     pthread_mutex_unlock(&dm_list_mutex);
3227:
3228:     //If new ip send all known xTEDS

```

```

3229:     if(NewAdd)
3230:     {
3231:         debug_f(3, "Backup added to the Backup DM List \n");
3232:         SDMxTEDS msgxTEDS;
3233:
3234:         xTEDSLibraryListNode* node = xTEDSList.head;
3235:         while(node!=NULL)
3236:         {
3237:             // Fill the xTEDS info into the message
3238:             node->data->inUse->Wait();
3239:             msgxTEDS.source = node->data->getComponentID();
3240:             PID = node->data->getPid();
3241:             msgxTEDS.setPID();
3242:             msgxTEDS.active = node->data->getActive();
3243:             strncpy(msgxTEDS.xTEDS,node->data->getxTEDS(), sizeof(msgxTEDS.xTEDS));
3244:             if(node->data->getSPANode() != NULL)
3245:                 strcpy(msgxTEDS.SPA_node,node->data->getSPANode());
3246:             node->data->inUse->Signal();
3247:
3248:             // Send the message
3249:             pthread_mutex_lock(&dm_list_mutex);
3250:             backupDMList.SendMessageTo(BackupId, msgxTEDS);
3251:             pthread_mutex_unlock(&dm_list_mutex);
3252:
3253:             node = node->next;
3254:         }
3255:     }
3256: #endif
3257: }
3258: else
3259:     msgReady.Send(id);
3260: MessageSent(&msgReady);
3261: }
3262:
3263: /*
3264: Description:
3265:     Handling publishing the services that the DM provides. See the DM xTEDS
3266:
3267: Input:
3268:     ipaddr - the ip address in dot notation
3269:     port - the port of the consumer

```

```

3270:      sID - generally the sensorID of the device in question
3271:      DataReplyId - an ID for the message ID
3272:      seq_num - a number unique to the application that requested the service for identify which
response belongs to what request
3273:
3274:  Output:
3275:      None
3276:
3277:  Changed:
3278:      None
3279:
3280: */
3281: void ServicePublish(const SDMComponent_ID& RequesterId, unsigned int sID, const
SDMMessage_ID& DataReplyId, int seq_num)
3282: {
3283:     xTEDSLibraryListNode* node;
3284:     SDMDData msgData;
3285:     SDMComponent_ID tempID;
3286:
3287:     msgData.source = DataManager;
3288:     msgData.msg_id = DataReplyId;
3289:     msgData.seq_num = seq_num;
3290:
3291:     if(DataReplyId == RPLY_CONVERTED_DEVICE_NAME)
3292:     {
3293:         node = MatchSID(sID);
3294:         if(node == NULL)
3295:         {
3296:             printf("Device name not found for the given SensorID. \n");
3297:             return;
3298:         }
3299:
3300:         PUT_ULONG(&msgData.msg[0],sID);
3301:         strncpy(&msgData.msg[4],node->data->getName(),strlen(node->data->getName()+1));
3302:         msgData.Send(RequesterId,85);
3303:         MessageSent(&msgData);
3304:         node->data->inUse->Signal();
3305:         return;
3306:     }
3307:     else if(DataReplyId == RPLY_CONVERTED_SPANODE) // Convert a Sensor ID to a
SPANodePath
3308:     {

```



```

3309:     node = MatchSID(sID);
3310:     if(node == NULL)
3311:     {
3312:         printf("Physical location information not found for the given SensorID \n");
3313:         return;
3314:     }
3315:     //
3316:     // Fill the Sensor ID and the SPANodePath
3317:     PUT_UINT(msgData.msg, sID);
3318:     strncpy(msgData.msg + sizeof(unsigned long), node->data->getSPANode(), 79);
3319:     msgData.msg[sizeof(unsigned long) + 79] = '\0';
3320:     node->data->inUse->Signal();
3321:
3322:     msgData.Send(RequesterId,84);
3323:     MessageSent(&msgData);
3324:     return;
3325: }
3326: else if(DataReplyId == RPLY_CONVERTED_IP)
3327: {
3328:     node = MatchSID(sID);
3329:     if(node==NULL)
3330:     {
3331:         printf("IP address and port not found for the given SensorID. \n");
3332:         return;
3333:     }
3334:
3335:     unsigned long temp = node->data->getSensorID();
3336:     unsigned short tempPort = node->data->getTargetPort();
3337:     PUT_ULONG(msgData.msg, temp);
3338:     temp = node->data->getAddress();
3339:     PUT_ULONG(msgData.msg+4, temp);
3340:     PUT_USHORT(msgData.msg+8,tempPort);
3341:     msgData.Send(RequesterId,10);
3342:     MessageSent(&msgData);
3343:     node->data->inUse->Signal();
3344:     return;
3345:
3346: }
3347: else if(DataReplyId == RPLY_RETURN_COMP_KEY)
3348: {
3349:     const char* NOKEY = "NOKEY";

```

```

3350:     debug_f(3,"Working on converting componentID to componentKey \n");
3351:
3352:     node = MatchSID(sID);
3353:     if(node==NULL)
3354:     {
3355:         printf("ComponentKey not found for the given SensorID. \n");
3356:         return;
3357:     }
3358:     tempID.Marshal(msgData.msg,0);
3359:     if(node->data->getComponentKey()!=NULL)
3360:         strcpy(msgData.msg,node->data->getComponentKey());
3361:     else
3362:         strcpy(msgData.msg,NOKEY);
3363:     msgData.Send(RequesterId,129);
3364:     MessageSent(&msgData);
3365:     node->data->inUse->Signal();
3366:     return;
3367: }
3368: else if(DataReplyId == RPLY_ERRORS)
3369: {
3370:     PUT_ULONG(msgData.msg,droppedxTEDS);
3371:     PUT_ULONG(msgData.msg+4,droppedCancelxTEDS);
3372:     msgData.Send(RequesterId,8);
3373:     MessageSent(&msgData);
3374:     return;
3375: }
3376: else
3377: {
3378:     node = xTEDSList.head;
3379:     while(node!=NULL)
3380:     {
3381:         node->data->inUse->Wait();
3382:         if(node->data->getAvailable() == false)
3383:         {
3384:             if (node->data->getComponentID().getAddress() == RequesterId.getAddress())
3385:             {
3386:                 if(sID == node->data->getPid()) //sID is actually the pid in the case of reply ID 9
3387:                 {
3388:                     long pID = node->data->getSensorID();
3389:                     PUT_ULONG(msgData.msg,sID);
3390:                     PUT_LONG(&msgData.msg[4],pID);

```

```

3391:                msgData.Send(RequesterId,8);
3392:                MessageSent(&msgData);
3393:                debug_f(1,"Sending sensor ID of %ld for PID %u to 0x%lx at port %hu \n",
pID, sID, RequesterId.getAddress(), RequesterId.getPort());
3394:                node->data->inUse->Signal();
3395:                return;
3396:            }
3397:        }
3398:    }
3399:    node->data->inUse->Signal();
3400:    node = node->next;
3401:}
3402:    printf("Unable to identify PID %u that sensor ID is wanted for \n",sID);
3403:}
3404:}
3405:
3406:/*
3407: Description:
3408:     Handling of the SDMTat message
3409:
3410: Input:
3411:     buf - the message in an array
3412:     size - the length of the message in bytes
3413:     ipaddr - the ip address in dot notation
3414:
3415: Output:
3416:     None
3417:
3418: Changed:
3419:     None
3420:
3421: */
3422: void TAT(char *buf, int size, const SDMComponent_ID& SenderId)
3423: {
3424:     SDMTat tat;
3425:     int n = 0, i = 0;
3426:     const unsigned int S_SIZE = xsize;
3427:     unsigned long sentips[S_SIZE];
3428:     int sentcount = 0;
3429:     xTEDSLibraryListNode* node;
3430:     SDMComponent_ID id;

```

```

3431:
3432: if(tat.Unmarshal(buf) < 0)
3433: {
3434:     printf("Invalid SDMTAT message! \n \n");
3435:     return;
3436: }
3437: MessageReceived(&tat);
3438:
3439: debug_f(1,"SDMTat    seconds:    %ld    useconds:    %ld    sensorID:    %ld
\n",tat.seconds,tat.useconds,tat.destination.getSensorID());
3440:
3441: if(tat.destination.getSensorID() == 0)
3442: {
3443:     //Send tat to each SM
3444:     node = xTEDSList.head;
3445:     while(node!=NULL)
3446:     {
3447:         node->data->inUse->Wait();
3448:         if(node->data->getHub() == 1)
3449:         {
3450:             for(i = 0; i < sentcount; i++)
3451:             {
3452:                 if(sentips[i] == node->data->getAddress())
3453:                 {
3454:                     node->data->inUse->Signal();
3455:                     break;
3456:                 }
3457:             }
3458:             if(i == sentcount)
3459:             {
3460:                 sentips[sentcount] = node->data->getAddress();
3461:                 id.setAddress(node->data->getAddress());
3462:                 id.setPort(PORT_SM);
3463:                 tat.Send(id);
3464:                 MessageSent(&tat);
3465:                 sentcount++;
3466:                 debug_f(1,"Sent tat of %ld %ld %ld to 0x%lx at port %hu \n", tat.seconds,
tat.useconds, tat.destination.getSensorID(), node->data->getAddress(),PORT_SM);
3467:             }
3468:         }
3469:         node->data->inUse->Signal();

```

```

3470:         node = node->next;
3471:     }
3472:     if(sentcount == 0)
3473:         printf("No SM found, tat could not be sent to any SM's \n");
3474: }
3475: else
3476: {
3477:     //Send tat to SM that controls the sensor
3478:     node = xTEDSList.head;
3479:     while(node!=NULL)    //Determine which xTED this Consume corresponds to
3480:     {
3481:         node->data->inUse->Wait();
3482:         if(node->data->getAvailable() != true)
3483:         {
3484:             if(tat.destination.getSensorID() == node->data->getSensorID())
3485:             {
3486:                 if(node->data->getActive() == true)
3487:                 {
3488:                     if(node->data->getHub() == 1)
3489:                     {
3490:                         debug_f(4,"Found matching ID in %d \n",n);
3491:                         break;
3492:                     }
3493:                     else
3494:                     {
3495:                         printf("SensorID:  %ld  does  not  correspond  to  a  sensor  \n
\n",tat.destination.getSensorID());
3496:                         node->data->inUse->Signal();
3497:                         return;
3498:                     }
3499:                 }
3500:             }
3501:         }
3502:         node->data->inUse->Signal();
3503:         node = node->next;
3504:     }
3505:     if(node == NULL)
3506:     {
3507:         printf("No matching sensorID found, tat can not be sent \n \n");
3508:         return;
3509:     }

```

```

3510:     tat.Send(node->data->getComponentID());
3511:     MessageSent(&tat);
3512:     debug_f(1,"Sent tat of %ld %ld %ld to 0x%lx at port %hu \n", tat.seconds, tat.useconds,
tat.destination.getSensorID(), node->data->getAddress(), PORT_SM);
3513:     node->data->inUse->Signal();
3514: }
3515: }
3516:
3517: /*
3518:  Description:
3519:      Intialization of the pipes for passing messages through
3520:
3521:  Input:
3522:      None
3523:
3524:  Output:
3525:      None
3526:
3527:  Changed:
3528:      None
3529:
3530: */
3531: void PipeInit()
3532: {
3533: #ifndef __VXWORKS__
3534:     if(pipe(alert) < 0)
3535:     {
3536:         if(debug >= 2)
3537:             fprintf(stderr, "pipe error \n"); _exit(0);        // IPC for Consume(),Service(),Cancel()
3538:     }
3539: #else
3540:     if(pipeDevCreate("/pipe/alert", 10, 10) != 0)
3541:     {
3542:         if(debug >= 1)
3543:             fprintf(stderr, "Pipe error \n");
3544:     }
3545:     alertPipe = open ("/pipe/alert", O_RDWR);
3546: #endif
3547: }
3548:
3549: /*

```

```

3550: Description:
3551:     Handling messages passed through the pipe
3552:
3553: Input:
3554:     arg - a pointer that is not used
3555:
3556: Output:
3557:     None
3558:
3559: Changed:
3560:     None
3561:
3562: */
3563: void* ChildFunctionCallProcess(void* arg)
3564: {
3565:     int size = 0;
3566:     char buf[BUFSIZE];    //Buffer for recieving incoming messages
3567:     char len[8];
3568:     int BytesSoFar = 0;
3569:     SDMComponent_ID SenderId;
3570:     char SenderIdBuf[10];
3571:     chdir("./Callprofile");
3572:
3573:     debug_f(3,"Child function call process running ... \n");
3574:
3575:     //
3576:     // Set up the IPC channel via TCP
3577:     //
3578:     // Open a TCP server socket and begin listening for connections
3579:     int TCPSock;           // TCP listen socket
3580:     if ((TCPSock = TCPpassive(PORT_DM_TEMP, 1)) == IP_SOCKET_INVALID)
3581:     {
3582:         printf("Error - ChildFunctionCallProcess::Could not bind port %hu \n",PORT_DM_TEMP);
3583:         return NULL;
3584:     }
3585:     sockaddr_in SIn;
3586:     int MsgRxSock = IP_SOCKET_INVALID;    // Socket for receiving messages from
UDPListenerProcess
3587:     bool Connected = false;
3588:
3589:     // We should only receive one connection request, but just to be safe, limit the address...

```

```

3590: while (!Connected)
3591: {
3592:     MsgRxSock = TCPaccept(TCPSock, &SIn);
3593:     if (SIn.sin_addr.s_addr == Address_DM || SIn.sin_addr.s_addr == L_LOCAL_HOST ||
Address_DM == L_LOCAL_HOST)
3594:         Connected = true;
3595: }
3596: TCPclose(TCPSock);
3597: //
3598: // Connection up, wait for messages
3599: int iStatus;
3600: bool bError = false;
3601: while(1)
3602: {
3603: #ifndef __VXWORKS__
3604:     while(read(alert[0], len, 1) > 0)
3605: #else
3606:     while(read(alertPipe, len, 1) > 0)
3607: #endif
3608:     {
3609:         iStatus = 0;
3610:         bError = false;
3611:         BytesSoFar = 0;
3612:         // Receive the Sender component ID
3613:         while (BytesSoFar < 10)
3614:         {
3615:             iStatus = TCPrecv(MsgRxSock, SenderIdBuf + BytesSoFar, sizeof(SenderIdBuf) -
BytesSoFar);
3616:             if (iStatus < -1)
3617:             {
3618:                 perror("TCPrecv");
3619:                 bError = true;
3620:                 break;
3621:             }
3622:             else if (iStatus == 0)
3623:             {
3624:                 bError = true;
3625:                 break;
3626:             }
3627:             else
3628:                 BytesSoFar += iStatus;

```



```

3629:     }
3630:     if (bError)
3631:         continue;
3632:     SenderId.Unmarshal(SenderIdBuf, 0);
3633:     //
3634:     // Get the message size
3635:     debug_f(4,"Receiving size: ");
3636:     BytesSoFar = 0;
3637:     while (BytesSoFar < 4)
3638:     {
3639:         iStatus = TCPrevc(MsgRxSock, &size + BytesSoFar, 4 - BytesSoFar); //Get the size
from pipe
3640:         if (iStatus < -1)
3641:         {
3642:             perror("TCPrevc");
3643:             bError = true;
3644:             break;
3645:         }
3646:         else if (iStatus == 0)
3647:         {
3648:             bError = true;
3649:             break;
3650:         }
3651:         else
3652:             BytesSoFar += iStatus;
3653:     }
3654:     if (bError)
3655:         continue;
3656:     debug_f(4, "%d \n",size);
3657:     //
3658:     // Get the rest of the message
3659:     BytesSoFar = 0;
3660:     while (BytesSoFar < size)
3661:     {
3662:         iStatus = TCPrevc(MsgRxSock, &buf + BytesSoFar, size - BytesSoFar);
3663:         if (iStatus < -1)
3664:         {
3665:             perror("TCPrevc");
3666:             bError = true;
3667:             break;
3668:         }

```

```

3669:         else if (iStatus == 0)
3670:         {
3671:             bError = true;
3672:             break;
3673:         }
3674:         else
3675:             BytesSoFar += iStatus;
3676:     }
3677:     if (bError)
3678:         continue;
3679:     //
3680:     // Check the command message byte and send to correct function
3681:     switch(buf[0])
3682:     {
3683:     case SDM_Consume:
3684:         Consume(buf, size, SenderId);
3685:         break;
3686:     case SDM_Cancel:
3687:         Cancel(buf, size, SenderId);
3688:         break;
3689:     case SDM_ReqReg:
3690:         ReqReg(buf, size, -1, SenderId, MOD_NOT_APPLICABLE);
3691:         break;
3692:     case SDM_Service:
3693:         Service(buf, size, SenderId);
3694:         break;
3695:     case SDM_Serreqst:
3696:         Serreqst(buf, size, SenderId);
3697:         break;
3698:     case SDM_ReqxTEDS:
3699:         ReqxTEDS(buf, size, SenderId);
3700:         break;
3701:     case SDM_Command:
3702:         Command(buf, size, SenderId);
3703:         break;
3704:     case SDM_Ready:
3705:         Ready(buf, size, SenderId);
3706:         break;
3707:     case SDM_Tat:
3708:         TAT(buf, size, SenderId);
3709:         break;

```

```

3710:         case SDM_Search:
3711:             Search(buf, size, SenderId, -1);
3712:             break;
3713:         case SDM_VarReq:
3714:             VarReq(buf, size, SenderId, -1);
3715:             break;
3716:         case SDM_TaskError:
3717:             TaskError(buf, size, SenderId);
3718:             break;
3719:         case SDM_Heartbeat:
3720: #ifdef PNP_BACKUP
3721:             //debug_f(1, "command received: Task Error \n");
3722:             QueueHeartbeat(buf, size);
3723:             break;
3724: #endif
3725:         default:
3726:             debug_f(0, "command received: Invalid command recieved is 0x%x \n", buf[0]);
3727:             break;
3728:     }
3729: }
3730: }
3731: exit(0);
3732: }
3733:
3734: /*
3735:  Description:
3736:      Prints the active and inactive xTEDS that are registered
3737:
3738:  Input:
3739:      None
3740:
3741:  Output:
3742:      None
3743:
3744:  Changed:
3745:      None
3746:
3747: */
3748: void PrintxTEDS()
3749: {
3750:     int p=0;

```

```

3751: int ncount = 1;
3752: xTEDSLibraryListNode* node;
3753: char* ted = NULL;
3754:
3755: if(xTEDSList.head != NULL) //If any xTEDS are still registered, should always be true due to
DM never deregistering its xTEDS
3756: {
3757:     /*Print publish names*/
3758:     if(debug >= 1)
3759:     {
3760:         printf("Publish names are ");
3761:         node = xTEDSList.head;
3762:         while(node!=NULL)
3763:         {
3764:             node->data->inUse->Wait();
3765:             if(node->data->getAvailable() == false)
3766:             {
3767:                 if(node->data->getName() != NULL)
3768:                 {
3769:                     if(p == 0)
3770:                     {
3771:                         printf("%s",node->data->xtedsTree->getDeviceName());
3772:                         p = 1;
3773:                     }
3774:                     else
3775:                     {
3776:                         if(node->data->getActive() == true)
3777:                             printf(", %s",node->data->getName());
3778:                         else
3779:                             printf(", *%s",node->data->getName());
3780:                     }
3781:                 }
3782:             }
3783:             node->data->inUse->Signal();
3784:             node = node->next;
3785:         }
3786:     }
3787:
3788:     /*Print xTEDS*/
3789:     if(debug >= 5)
3790:     {

```

```

3791:     printf("xTEDs are: \n");
3792:     node = xTEDSList.head;
3793:     while(node!=NULL)
3794:     {
3795:         node->data->inUse->Wait();
3796:         if(node->data->getAvailable() == false)
3797:         {
3798:             if(node->data->getxTEDS() != NULL)
3799:             {
3800:                 if(ted!=NULL) free(ted);
3801:                 ted = strdup(node->data->getxTEDS());
3802:                 if(node->data->getActive() == true)
3803:                     printf("%d.  %s",ncount,ted);
3804:                 else
3805:                     printf("%d.* %s",ncount,ted);
3806:                 ncount++;
3807:                 if(ted[strlen(ted)-1] != '\n')
3808:                     printf(" \n");
3809:             }
3810:         }
3811:         node->data->inUse->Signal();
3812:         node = node->next;
3813:     }
3814: }
3815: }
3816: if(ted != NULL) free(ted);
3817: debug_f(1," \n \n");
3818: }
3819:
3820: /*
3821:  Description:
3822:      Check to see if the xTEDS is already registered
3823:
3824:  Input:
3825:      sensor - the name of the sensor from the xTEDS
3826:      buf - the message in an array
3827:      port - the port device is using to communicate on
3828:      addr - the ip addr in dot notation
3829:      type - 0 for application, 1 for device, 2 for robohub
3830:      spot - a reference index for the node that contains the xTEDS of the device if already
registered

```

```

3831:
3832:  Output:
3833:      xTEDSLibraryListNode* - a pointer to the xTEDSLibraryListNode for the xTEDS
3834:
3835:  Changed:
3836:      None
3837:
3838: */
3839: xTEDSLibraryListNode* AlreadyRegistered(char* sensor, const char* buf, unsigned short port,
SDMComponent_ID& idAddr, int type, int& spot)
3840: {
3841:     int temp = 0, count = 0;
3842:     long ID;
3843:     long tempID = 0;
3844:     xTEDSLibraryListNode* node;
3845:     SDMxTEDS ted;
3846:
3847:     ted.Unmarshal(buf);
3848:     node = xTEDSList.head;
3849:     while(node != NULL)    //Check to see if the xted is registered already as inactive
3850:     {
3851:         node->data->inUse->Wait();
3852:         if(node->data->getAvailable() == false)
3853:         {
3854:             if(strncmp(node->data->getName(), sensor, strlen(sensor)) == 0)
3855:             {
3856:                 if(node->data->getActive() == false)
3857:                 {
3858:                     node->data->setActive(true);
3859:                     node->data->setAddress(idAddr);
3860:
3861:                     ID = ted.source.getSensorID();
3862:                     node->data->setTargetPort(ted.source.getPort());
3863:                     node->data->setPid(ted.getPID());
3864:
3865:                     node->data->setHub(type);
3866:                     if(strncmp(node->data->getName(), "RoboHub", 7) == 0 || strncmp(node->data-
>getName(), "RobustHub", 9) == 0)
3867:                     {
3868:                         debug_f(3, "Robust Hub being registered \n");
3869:                         node->data->setHub(ROBOHUB);

```

```

3870:         }
3871:
3872:         //Create SensorID
3873:         debug_f(4,"ID is %ld \n",ID);
3874:         ID &= 0x0000FFFF;    //Create the unique ID
3875:         //if(ID == 0) // cbj -- Removed, ASIMs don't set this byte when the need to re-
register, I don't know
3876:             //node->data->setActive(false); // when this has ever been used, the removal
should be OK.
3877:         debug_f(4,"ID after the and with 0x000000FF is %ld \n",ID);
3878:         temp = count<<16;
3879:         debug_f(4,"spot %d shifted by 16 is %hd \n",count,temp);
3880:         ID |= (count<<16);
3881:         debug_f(4,"ID after or is %ld \n",ID);
3882:         node->data->setSensorID(ID);
3883:         debug_f(1,"SensorID for %s will be %ld \n",sensor,node->data->getSensorID());
3884:         spot = count;
3885:
3886:         node->data->inUse->Signal();
3887:         return node;
3888:     }
3889: else if(port == node->data->getTargetPort())
3890: {
3891:     if (idAddr.getAddress() == node->data->getAddress())
3892:     {
3893:         ID = ted.source.getSensorID(); //Copy the id from buf into local variable
3894:         ID &= 0x0000FFFF;    //Create the unique ID
3895:         tempID = node->data->getSensorID(); //Get sensor ID DM created
3896:         tempID &= 0x0000FFFF;    //Change DM sensor ID to ID assigned by SM
or application
3897:         if(ID == tempID && port != PORT_TM)
3898:         {
3899:             printf("Device already registered \n \n");
3900:             spot = -1;
3901:             node->data->inUse->Signal();
3902:             return node;
3903:         }
3904:         if(port == PORT_TM)
3905:         {
3906:             node->data->inUse->Signal();
3907:             spot = count;
3908:             return node;

```

```

3909:         }
3910:     }
3911: }
3912: }
3913: }
3914:     node->data->inUse->Signal();
3915:     node = node->next;
3916:     count++;
3917: }
3918: debug_f(5,"Unable to find a previously registered xTED \n");
3919: return NULL;
3920: }
3921:
3922: /*
3923:  Description:
3924:      Handling of the SDMSearch message
3925:
3926:  Input:
3927:      sensor - the name of the sensor from the xTEDS
3928:      spot - a reference index for the node the xTEDS has been stored in
3929:      addr - the ip address of the device in dot notation
3930:      buf - the message in an array
3931:      type - 0 for application, 1 for device, 2 for robohub
3932:
3933:  Output:
3934:      xTEDSLibraryListNode* - a pointer to the xTEDSLibraryListNode the xTEDS has been
stored in
3935:
3936:  Changed:
3937:      None
3938:
3939: */
3940: xTEDSLibraryListNode* StoreInfo(char* sensor, int &spot, const SDMComponent_ID& idAddr,
const char* buf, int type)
3941: {
3942:     unsigned long ID;
3943:     int count = 0;
3944:     xTEDSLibraryListNode* node = NULL;
3945:     xTEDSLibraryListNode* iterator;
3946:     SDMxTEDS ted;
3947:     int active;

```



```

3948: char SPAHub[128];
3949:
3950: if(spot > 0 || xTEDSList.head != NULL)
3951: {
3952:     iterator = xTEDSList.head;
3953:     do
3954:     {
3955:         count++;
3956:         iterator = iterator->next;
3957:     }while(iterator!=NULL && count!=spot && iterator->data->getAvailable()!=true);
3958:     node = iterator;
3959: }
3960: if(node == NULL)
3961: {
3962:     xTEDSLibrary* lib;
3963:     lib = new xTEDSLibrary();
3964:     xTEDSList.addLibrary(lib);
3965:     node = xTEDSList.tail;
3966: }
3967:
3968: ted.Unmarshal(buf);
3969: node->data->inUse->Wait();
3970: node->data->setAddress(idAddr);
3971: node->data->setAvailable(false);
3972:
3973: ID = ted.source.getSensorID();           //Copy the id from buf into local variable
3974: node->data->setTargetPort(ted.source.getPort()); //Save the port to communicate on
3975: node->data->setPid(ted.getPID());
3976: node->data->setSPANode(ted.SPA_node);
3977:
3978: node->data->setHub(type);
3979: if(node->data->getHub() == ROBOHUB)
3980: {
3981:     if(GetSPAHub(ted.xTEDS, SPAHub, debug)==0)
3982:         node->data->setSPAHub(SPAHub);
3983: }
3984: debug_f(4, "Type of device is %d \n", node->data->getHub());
3985: //Create SensorID
3986: debug_f(4, "ID is %ld \n", ID);
3987: ID &= 0x000000FF;    //Create the unique ID
3988: active = ted.active;

```

```

3989:  if(active == 0)
3990:      node->data->setActive(false);
3991:  else
3992:      node->data->setActive(true);
3993:  debug_f(4,"ID after the and with 0x000000FF is %ld \n",ID);
3994:  debug_f(4,"spot %hd shifted by 16 is %hd \n",count,(count<<16));
3995:  ID |= (count<<16);
3996:  debug_f(4,"ID after or is %ld \n",ID);
3997:  node->data->setSensorID(ID);
3998:  debug_f(1,"SensorID for %s will be %ld \n",sensor,node->data->getSensorID());
3999:  debug_f(4,"SM connected from 0x%lx on port %hu \n",node->data->getAddress(),node->data->getTargetPort());
4000:  spot = count;
4001:  return node;
4002: }
4003:
4004: /*
4005:  Description:
4006:      Handling of sending a SDMPPostTask message to the Task Manager
4007:
4008:  Input:
4009:      node - a pointer to the xTEDSLibraryListNode node in question
4010:
4011:  Output:
4012:      None
4013:
4014:  Changed:
4015:      None
4016:
4017: */
4018: void PostTask(xTEDSLibraryListNode* node)
4019: {
4020:     SDMPPostTask task;
4021:     short resource;
4022:
4023:     if(TaskManager.getAddress()==0)
4024:     {
4025:         printf("Unable to post task because no TM is registered!!! \n");
4026:         return;
4027:     }
4028:     //This is a first test of posting Tasks and will be updated to get info from xTED for applications

```

```

4029: #ifdef __uClinux__
4030:  resource = SDM_MICROBLAZE | SDM_MEM128 | SDM_LINUX26;
4031: #elif WIN32
4032:  resource = SDM_INTEL|SDM_MEM128|SDM_WIN32;
4033: #else
4034:  //
4035:  // In the future, DM will probably have to hang on to a task's resource definitions
4036:  // This won't work if there are heterogeneous processors being used
4037:  resource = SDM_INTEL|SDM_MEM128|SDM_LINUX26;
4038: #endif
4039:  task.resources = resource;
4040:  task.priority = 1;
4041:  const char* name = node->data->getName();
4042:  memcpy(task.filename,name,strlen(name)+1);
4043:  debug_f(2,"Posting Task with resources of %d priority of %d and filename of %s\n",task.resources,task.priority,task.filename);
4044:  task.Send();
4045:  MessageSent(&task);
4046:  debug_f(2,"Task Posted \n");
4047: }
4048:
4049: /*
4050:  * Separate thread for signal handling. This avoids a potential deadlock situation in which the
  ChildFunctionCallProcess function
4051:  * is chosen to handle the SIGALRM signal after it has called
  pthread_mutex_lock(&subscription_mutex), which is also called in the
4052:  * signal handler.
4053:  */
4054: #ifndef WIN32
4055: void* SigHandler(void *arg)
4056: {
4057:  sigset_t signal_set;
4058:  int sig;
4059:  sigemptyset(&signal_set);
4060:  sigaddset(&signal_set, SIGINT);
4061:  sigaddset(&signal_set, SIGALRM);
4062:
4063:  while (1)
4064:  {
4065:    sigwait(&signal_set, &sig);
4066:    switch(sig)
4067:    {

```

```

4068:         case SIGINT:
4069:             printf("Shutting down process \n");
4070:             if ( udpSock != IP_SOCKET_INVALID )
4071:                 UDPshutdown(udpSock);
4072:             if ( tcpListenSock != IP_SOCKET_INVALID )
4073:                 TCPshutdown(tcpListenSock);
4074:             if ( tcpSock != IP_SOCKET_INVALID )
4075:                 TCPshutdown(tcpSock);
4076:             if ( MsgTxPushSock != IP_SOCKET_INVALID )
4077:                 TCPshutdown(MsgTxPushSock);
4078: #ifdef __VXWORKS__
4079:             close(alertPipe);
4080:             pipeDevDelete("/pipe/alert", true);
4081: #endif
4082:             exit(EXIT_SUCCESS);
4083:             break;
4084:         case SIGALRM:             //Publish performance counter message
4085:             PublishNotification(0, NOTI_PERFORMANCE_COUNTERS);
4086:             pthread_mutex_lock(&perf_counter_mutex);
4087:             prevsec_recd = 0;
4088:             prevsec_sent = 0;
4089:             pthread_mutex_unlock(&perf_counter_mutex);
4090:             break;
4091:         default:
4092:             printf("Signal: %i received \n", sig);
4093:     }
4094: }
4095: }
4096: #endif
4097: #ifdef WIN32
4098: void SigHandler(int sig)
4099: {
4100:     switch (sig)
4101:     {
4102:         case SIGINT:
4103:             printf("Shutting down process \n");
4104:             if ( udpSock != 0 )
4105:                 UDPshutdown(udpSock);
4106:             if ( tcpListenSock != 0 )
4107:                 TCPshutdown(tcpListenSock);
4108:             if ( tcpSock != 0 )

```

```

4109:         TCPshutdown(tcpSock);
4110:     if ( MsgTxPushSock != 0 )
4111:         TCPshutdown(MsgTxPushSock);
4112:     exit(EXIT_SUCCESS);
4113: case SIGALRM:     //Publish performance counter message
4114:     PublishNotification(0, NOTI_PERFORMANCE_COUNTERS);
4115:     pthread_mutex_lock(&perf_counter_mutex);
4116:     prevsec_recd = 0;
4117:     prevsec_sent = 0;
4118:     pthread_mutex_unlock(&perf_counter_mutex);
4119:     break;
4120: }
4121: }
4122: #endif
4123:
4124: //Function to increment performance counters and alert the log service of a message sent, making
the code simpler, smaller, and more readable.
4125: void MessageSent(SDMmessage *msg)
4126: {
4127:     pthread_mutex_lock(&perf_counter_mutex);
4128:     total_sent++;
4129:     prevsec_sent++;
4130:     pthread_mutex_unlock(&perf_counter_mutex);
4131:
4132:     pthread_mutex_lock(&log_service_mutex);
4133:     if (!log_service.IsEmpty())
4134:         log_service.MessageSent(msg);
4135:     pthread_mutex_unlock(&log_service_mutex);
4136:
4137: }
4138:
4139: //Function to log, but not count, the messages received. This makes the code simpler, smaller, and
more readable.
4140: void MessageReceived(SDMmessage *msg)
4141: {
4142:     pthread_mutex_lock(&log_service_mutex);
4143:     if (!log_service.IsEmpty())
4144:         log_service.MessageReceived(msg);
4145:     pthread_mutex_unlock(&log_service_mutex);
4146:
4147: }

```

```

4148:
4149: /*
4150:  Description:
4151:      Find an xTEDS node that has a matching sensor ID
4152:
4153:  Input:
4154:      sID - the sensor ID to be matched
4155:
4156:  Output:
4157:      xTEDSLibraryListNode* - a pointer to the node that has the sID
4158:
4159:  Changed:
4160:      None
4161:
4162: */
4163: xTEDSLibraryListNode* MatchSID(long sID)
4164: {
4165:     xTEDSLibraryListNode* node = xTEDSList.head;
4166:
4167:     while(node!=NULL)
4168:     {
4169:         node->data->inUse->Wait();
4170:         if(node->data->getActive() == true)
4171:         {
4172:             if(((unsigned long)sID == node->data->getSensorID()))
4173:             {
4174:                 return node;//Lock must be released in calling function
4175:             }
4176:         }
4177:         node->data->inUse->Signal();
4178:         node = node->next;
4179:     }
4180:     return NULL;
4181: }
4182:
4183: /*
4184:  Description:
4185:      Handling of sending xTEDS to the backup DM's
4186:
4187:  Input:
4188:      arg - a pointer to the xTEDSPParameters

```

```

4189:
4190: Output:
4191:     None
4192:
4193: Changed:
4194:     None
4195:
4196: */
4197: void* SendxTEDS(void* arg)
4198: {
4199:     int size;
4200:     xTEDSPParameters* param;
4201:     SDMxTEDS xTEDS;
4202:     SDMComponent_ID id;
4203:
4204:     param = (xTEDSPParameters*) arg;
4205:     const char* buf = param->getBuffer();
4206:     size = param->getSize();
4207:
4208:     if(xTEDS.Unmarshal(buf) < 0)
4209:     {
4210:         printf("Error sending xTEDS to backup DM! \n \n");
4211:         return 0;
4212:     }
4213:     id = param->getSenderAddress();
4214:     id.setPort(PORT_DM_ELECTION);
4215:
4216:     xTEDS.SendTo(id);
4217:     return 0;
4218: }
4219: #ifdef BUILD_FOR_XTEDS_MERGING
4220: /*
4221: Description:
4222:     Handling of merging sdm.config file information with xTEDS
4223:
4224: Input:
4225:     flag - a pointer to the node that is to be merged into after a robohub has registered
4226:     update - Check to see if an unmerged xTEDS can be merged after the registration of a
4227:     robohub
4228:     ted - the SDMxTEDS message
4229:     xTEDref - the index of the xTEDSLibraryListNode

```

```

4229:      HubType - 0 for applications, 1 for devices, 2 for robohubs
4230:
4231:  Output:
4232:      Returns whether the merge was successful
4233:
4234:  Changed:
4235:      None
4236:
4237: */
4238: bool MergexTEDS(xTEDSLibraryListNode* flag, bool update, SDMxTEDS& ted, int xTEDref,
int HubType)
4239: {
4240:  char SPAHub[128] = "";
4241:  char SPAPort[128] = "";
4242:  char HubAddress[MAX_USB_PATH_SIZE] = "";
4243:  char config[BUFSIZE];
4244:  int temp = 0, count;
4245:  xTEDSLibraryListNode* node;
4246:  char oldxTEDS[3 * BUFSIZE];
4247:  unsigned long SID = 0;
4248:  bool active = false;
4249:
4250:  //Be sure that we aren't dealing with an application
4251:  if(HubType > APPLICATION)    // device or hub
4252:  {
4253:      //Get the string length of the USB path
4254:      temp = strlen(ted.SPA_node);
4255:      debug_f(4, "Length of USB address string is %d (USB address is %s)
\n", temp, ted.SPA_node);
4256:
4257:      //If the USB path contains something
4258:      if(temp > 0)
4259:      {
4260:          debug_f(3, "USB address registered %s \n", ted.SPA_node);
4261:          //If this is a device (not a robohub)
4262:          if(HubType == DEVICE)
4263:          {
4264:              //Find this device's hub's address in the config file
4265:              if (!FindDevicesHubPath(ted.SPA_node, HubAddress, sizeof(HubAddress), debug))
4266:              {
4267:                  debug_f(1, "Could not find hub information for device at %s in configuration file.
\n", ted.SPA_node);

```



```

4268:         return false;
4269:     }
4270:     else
4271:         debug_f(3,"Hub path found in configuration file is %s. \n",HubAddress);
4272: }
4273: temp = -1;
4274: if(HubType == ROBOHUB)
4275:     temp = 0;
4276: count = 0;
4277: node = xTEDSList.head;
4278: while(node!=NULL)        // search through the registered items looking for the hub....
4279: {
4280:     char* SavedPortAddress = NULL;
4281:     node->data->inUse->Wait();
4282:     //If this item is a hub and it is active
4283:     if(node->data->getHub() == ROBOHUB && node->data->getActive() == true)
4284:     {
4285:         //Pull out its saved SavedHubHumber
4286:         SavedPortAddress = node->data->getSPANode();
4287:         debug_f(4,"Found hub with address %s in location %d trying to match %s \n",
SavedPortAddress, count, HubAddress);
4288:
4289:         //If the device being registered is connected to a previously registered RoboHub
4290:         if(SavedPortAddress!=NULL && strcmp(SavedPortAddress, HubAddress) == 0)
4291:         {
4292:             debug_f(4,"Found matching hub in location %d \n",count);
4293:             //Save its position in the xTEDS list
4294:             temp = count;
4295:             break;
4296:         }
4297:     }
4298:     node->data->inUse->Signal();
4299:     count++;
4300:     node = node->next;
4301: }
4302: if(debug >= 3)
4303: {
4304:     if(node==NULL)
4305:         debug_f(3,"No matching hub found... \n");
4306:     else

```

```

4307:          debug_f(3,"ConvertToHubAxis  returned  %d  which  corresponds  to  %s
\n",temp,node->data->getSPAHub());
4308:      }
4309:      //If this is a device whose Robohub has been found, or a Robohub being registered
4310:      if(temp != -1)
4311:      {
4312:          if(HubType == DEVICE)
4313:          {
4314:              strcpy(SPAPort, ted.SPA_node);
4315:              strcpy(SPAHub, HubAddress);
4316:          }
4317:          else//ROBOHUB
4318:              strcpy(SPAHub,ted.SPA_node);
4319:          //See if the config information about the device exists in the configuration file we
will try to merge it
4320:          if(FindConfigInfo("Device",config,SPAHub,SPAPort,debug) != -1)
4321:          {
4322:              //It did exist, now attempt the merge
4323:              if(MergeConfigxTED(ted.xTEDS,sizeof(ted.xTEDS),config,debug) == 0)
4324:              {
4325:                  debug_f(3,"Merge succeeded new xTED is %s \n",ted.xTEDS);
4326:                  if(update == false)
4327:                  {
4328:                      if(node != NULL)
4329:                          node->data->inUse->Signal();
4330:                      return true;
4331:                  }
4332:              }
4333:              else
4334:              {
4335:                  printf("Merge failed!! \n");
4336:                  update = false;
4337:              }
4338:          }
4339:          //Device not found in the configuration file
4340:          else
4341:          {
4342:              if(node != NULL)
4343:                  printf("Could not match any info to merge with for hub: %s and port: %s
\n",node->data->getSPAHub(),ted.SPA_node);
4344:              update = false;
4345:          }

```

```

4346:         if(node != NULL)
4347:             node->data->inUse->Signal();
4348:     }
4349:     //A device's robohub has not been registered
4350:     else
4351:     {
4352:         printf("Could not match device at %s with any Robust Hub. \n",ted.SPA_node);
4353:         update = false;
4354:     }
4355: }
4356: //Length of USB path was empty
4357: else
4358: {
4359:     printf("USB address for device was not sent in the msg! \n");
4360:     update = false;
4361: }
4362: }
4363: //
4364: //If the update succeeded, alert any interested subscribers about the xTEDS
4365: if(update == true)
4366: {
4367:     if(flag == NULL)
4368:         return false;
4369:     //Make change to xTEDS and parse
4370:     const char* xTEDS = flag->data->getXTEDS();
4371:     memcpy(oldxTEDS,xTEDS,strlen(xTEDS) + 1);
4372:
4373:     //If parse fails reparse old xTEDS and send ack with error
4374:     if(flag->data->setxTEDS(ted.xTEDS) == false)
4375:     {
4376:         if(flag->data->setxTEDS(oldxTEDS) == false)
4377:         {
4378:             printf("Data Manager has been corrupted!!!! \n");
4379:         }
4380:         else
4381:             printf("Reverting back to the old xTEDS due to the merge corrupting the xTEDS.
\n");
4382:     }
4383:     else
4384:     {
4385:         flag->data->inUse->Wait();

```

```

4386:         active = flag->data->getActive();
4387:         SID = flag->data->getSensorID();
4388:         flag->data->inUse->Signal();
4389:
4390:         debug_f(4,"Checking for any subscriptions \n");
4391:         if (active)
4392:         {
4393:             PublishNotification(SID, NOTI_XTEDS_MODIFICATION);
4394:             PublishNotification(SID, NOTI_REGISTRATION_CHANGE,
UPDATE_MODIFICATION);
4395:             PublishxTEDSMModificationSubscription(SID, SEARCH_REPLY,
MOD_NOT_APPLICABLE, xTEDref);
4396:             PublishxTEDSMModificationSubscription(SID, REQ_REG_FUTURE,
MOD_NOT_APPLICABLE, xTEDref);
4397:             PublishxTEDSMModificationSubscription(SID, VAR_REQ_REPLY,
MOD_NOT_APPLICABLE, xTEDref);
4398:         }
4399:     }
4400: }
4401: return update;
4402: }
4403: #endif // #ifdef BUILD_FOR_XTEDS_MERGING
4404: /*
4405: Description:
4406:     Handling of SDMVarReq messages
4407:
4408: Input:
4409:     buf - a pointer to the array containing the message
4410:     size - the length of the message
4411:     ipaddr - the ip address of the sender in dot notation
4412:     xTEDref - the index of the xTEDSLibraryListNode
4413:
4414: Output:
4415:     None
4416:
4417: Changed:
4418:     None
4419:
4420: */
4421: void VarReq(char* buf, int size, const SDMComponent_ID& SenderId, int xTEDref)
4422: {
4423:     xTEDSLibraryListNode* node;

```

```

4424: SDMVarReq msgVarRequest;
4425: SDMVarInfo msgVarInfo;
4426: SDMComponent_ID ReplyId;
4427: VariableDef* msg = NULL;
4428: VariableDef* cur_msg = NULL;
4429: int ref = -1;
4430:
4431: if(msgVarRequest.Unmarshal(buf) < 0)
4432: {
4433:     printf("Invalid SDMSearch message \n \n");
4434:     return;
4435: }
4436: MessageReceived(&msgVarRequest);
4437:
4438: //
4439: // Validate the requester's port
4440: if(msgVarRequest.destination.getPort() == 0)
4441: {
4442:     printf("Unable to complete SDMVarReq because of invalid destination port 0. \n");
4443:     return;
4444: }
4445: // Set the destination address if not known
4446: ReplyId = msgVarRequest.destination;
4447: if(ReplyId.getAddress() == 0)
4448:     ReplyId.setAddress(SenderId.getAddress());
4449:
4450: // Set the source address if not known
4451: if(msgVarRequest.source.getAddress() == L_LOCAL_HOST && Address_DM !=
L_LOCAL_HOST)
4452:     msgVarRequest.source.setAddress(Address_DM);
4453:
4454: if (debug >= 1)
4455: {
4456:     char strRequesterId[64];
4457:     char strTargetId[64];
4458:     msgVarRequest.source.IDToString(strTargetId, sizeof(strTargetId));
4459:     msgVarRequest.destination.IDToString(strRequesterId, sizeof(strRequesterId));
4460:     debug_f(1, "SDMVarReq from %s for %s variable: \"%s\" \n",
4461:         strRequesterId, strTargetId, msgVarRequest.variable);
4462: }
4463:

```

```

4464: //
4465: // Add a subscription entry if entered
4466: if(msgVarRequest.reply != SDM_VARREQ_CURRENT)
4467: {
4468:     char interface_id[8] = "";
4469:     if(msgVarRequest.msg_id.getInterfaceMessagePair() != 0 || msgVarRequest.reply !=
SDM_VARREQ_CANCEL)
4470:         sprintf(interface_id,"%d",msgVarRequest.msg_id.getInterfaceMessagePair());
4471:
4472:     pthread_mutex_lock(&subscription_mutex);
4473:     bool result = subscribers.addOrRemoveSubscription(msgVarRequest.reply, ReplyId,
msgVarRequest.source, NULL, interface_id, msgVarRequest.variable, NULL, msgVarRequest.id,
VAR_REQ_REPLY, debug);
4474:     pthread_mutex_unlock(&subscription_mutex);
4475:     if(result == false)
4476:     {
4477:         return;
4478:     }
4479: #ifdef PNP_BACKUP
4480:     // Send the VarReq subscription to all backup DMs
4481:     pthread_mutex_lock(&dm_list_mutex);
4482:     msgVarRequest.destination = ReplyId;
4483:     backupDMList.SendMessageToAll(msgVarRequest);
4484:     pthread_mutex_unlock(&dm_list_mutex);
4485: #endif
4486: }
4487:
4488: node = xTEDSList.head;
4489: if(xTEDref != -1)
4490: {
4491:     int count = 0;
4492:     while(xTEDref != count && node != NULL)
4493:     {
4494:         count++;
4495:         node = node->next;
4496:     }
4497:     if(node == NULL)
4498:     {
4499:         printf("Unable to do VarReq due to invalid xTEDS reference \n");
4500:         return;
4501:     }
4502: }

```

```

4503: else
4504: {
4505:     if(msgVarRequest.source.getAddress() != 0)
4506:     {
4507:         if(msgVarRequest.source.getPort() != 0)
4508:         {
4509:             if(msgVarRequest.source.getSensorID() != 0)
4510:             {
4511:                 while(node!=NULL)
4512:                 {
4513:                     if(node->data->getComponentID()==msgVarRequest.source)
4514:                     {
4515:                         ref = 1;
4516:                         break;
4517:                     }
4518:                     node = node->next;
4519:                 }
4520:             }
4521:         }
4522:     }
4523: }
4524: while(node!=NULL)
4525: {
4526:     if(node->data->inUse->Wait() == 0)
4527:     {
4528:         if(node->data->getActive() == true)
4529:         {
4530:             debug_f(3,"Found matching xTEDS \n");
4531:
4532:             msg          =          node->data->xtedsTree->getVarInfo(msgVarRequest.variable,
msgVarRequest.msg_id);
4533:             cur_msg = msg;
4534:             while(cur_msg != NULL)
4535:             {
4536:                 // Fill the reply message
4537:                 msgVarInfo.source = node->data->getComponentID();
4538:                 msgVarInfo.id = msgVarRequest.id;
4539:                 strcpy(msgVarInfo.var_xTEDS,cur_msg->GetDefinitions());
4540:                 strcpy(msgVarInfo.interface,cur_msg->GetInterfaceName());
4541:                 // Send the reply message
4542:                 msgVarInfo.Send(ReplyId);

```

```

4543:         MessageSent(&msgVarInfo);
4544:         if (debug >= 1)
4545:         {
4546:             char strTargetAddr[64];
4547:             msgVarRequest.destination.IDToString(strTargetAddr, sizeof(strTargetAddr));
4548:             debug_f(1, " Send SDMVarInfo to %s var_xTEDS:  \"%s \" \n",
strTargetAddr, msgVarInfo.var_xTEDS);
4549:         }
4550:
4551:         cur_msg = cur_msg->next;
4552:     }
4553:     if(msg != NULL)
4554:         delete msg;
4555: }
4556:     node->data->inUse->Signal();
4557: }
4558: if(ref != -1)
4559:     break;
4560:     node = node->next;
4561: }
4562: // Send the SDMVarInfo stream terminator message
4563: msgVarInfo.SendEmpty(ReplyId);
4564: MessageSent(&msgVarInfo);
4565: }
4566:
4567: //////////////////////////////////////
4568: //
4569: //     The below functions pertain to the Data Manager backup feature.
4570: //
4571: //////////////////////////////////////
4572: /*
4573:  * Certain operations should only be performed if we are currently the elected DM.  If Backup is
not defined,
4574:  * this function always returns true.
4575:  */
4576: bool IAmElected()
4577: {
4578: #ifndef PNP_BACKUP
4579:     return true;
4580: #else
4581:     pthread_mutex_lock(&dm_list_mutex);

```



```

4582:  bool bResult = g_bElectedDm;
4583:  pthread_mutex_unlock(&dm_list_mutex);
4584:  return bResult;
4585: #endif
4586: }
4587:
4588:
4589:
4590: #ifdef PNP_BACKUP
4591:
4592: //*****
4593: //Description: Starts an election by callin the NMElection function. If elected the DM
4594: // will listen on Port 3505 for any other election related messages. If not elected,
4595: // the NonElected function is called and does not return until another election is
4596: // needed.
4597: //Input: None
4598: //Output: None
4599: //*****
4600:
4601: int RunBackupListener(void)
4602: {
4603:     SDMDMLeader templeader;
4604:     bool started = true;
4605:     char buf[BUFSIZE];
4606:     bool sent = false;
4607:     MessageManager mm;
4608:     pthread_attr_t attr;
4609:     pthread_t udpThread;
4610:     //Start the Message Manager listening on the DM Election port for both UDP and TCP
4611:     if(!mm.Async_Init_Both(PORT_DM_ELECTION))
4612:     {
4613:         printf("Message Manager not able to start up one of the listening interfaces! \n");
4614:         return -1;
4615:     }
4616:     NMElection(&mm); //Initiates the election process with the Network Manager
4617:     while(1)
4618:     {
4619:

```

```

4620:     if(g_bElectedDm == true) //This is the elected DM
4621:     {
4622:         pthread_attr_init(&attr);
4623:         pthread_attr_setstacksize(&attr,DM_STACK_SIZE);
4624:         DataManager.setAddress(GetNodeAddress());
4625:         if (0 != pthread_create(&udpThread,&attr,UdpListenerProcess,NULL))
4626:         {
4627:             perror("Could not start the UDP listener thread. \n");
4628:             return -1;
4629:         }
4630:         pthread_detach(udpThread);
4631:
4632:         while(1)
4633:         {
4634:             mm.BlockGetMessage(buf);
4635:             switch(buf[0])
4636:             {
4637:                 case SDM_DMLeader:
4638:                     templeader.Unmarshal(buf);
4639:                     debug_f(3,"Received SDMDMLeader message while running with
running_flag of %x \n",templeader.running_flag);
4640:                     break;
4641:                 case SDM_Election:
4642:                     //Clean up because this Data Manager has a problem and a new DM is being
elected.
4643:                     printf("Data Manager is shutting down. \n");
4644:                     return -1;
4645:                     break;
4646:                 case SDM_Ready:
4647:                     break;
4648:                 default:
4649:                     debug_f(1,"Unexpected message of type %c on PORT_DM_ELECTION
\n",buf[0]);
4650:                     break;
4651:             }
4652:         }
4653:     }
4654:     else //This is a backup DM
4655:     {
4656:         NonElected(&mm); //Returns when new election is needed
4657:     }
4658:     NMElection(&mm); //New Election Needed

```

```

4659: }
4660: }
4661:
4662:
//*****
****
4663: //Description: Sends an IMA message to the Network Manager and then calls the AmIPrimary
4664: // function which returns once the election has been resolved. The HeartBeat thread is
4665: // then started up.
4666: //Input: MessageManger* - A pointer to the message managaer used in the RunBackupListener
4667: // function.
4668: //Output: Global g_bElectedDm variable is set
4669:
//*****
****
4670: void NMElection(MessageManager* mm)
4671: {
4672:     if(spacewire)
4673:     {
4674: #ifdef SEND_IMA
4675:         SendIMA(SdmImaDm, debug);
4676: #endif
4677:     }
4678:     else
4679:     {
4680: #ifdef PNP_FAKE
4681:         DMSendIMA(); //Sends fake IMA to fake NM
4682: #endif
4683:     }
4684:     pthread_mutex_lock(&dm_list_mutex);
4685:     g_bElectedDm = AmIPrimary(mm); //Resolve Election
4686:     pthread_mutex_unlock(&dm_list_mutex);
4687:
4688:     pthread_t HeartbeatThread;
4689:     pthread_attr_t attr;
4690:     pthread_attr_init(&attr);
4691:     pthread_attr_setstacksize(&attr,DM_STACK_SIZE);
4692:     pipe(dm_heartbeat_pipe);
4693:     if (0 != pthread_create(&HeartbeatThread, &attr, &Heartbeat, NULL))
4694:     {
4695:         perror("Could not create the DM heartbeat thread. \n");
4696:         return;

```

```

4697: }
4698: }
4699:
4700: //Temporary function to fake an IMA message to our fake NM
4701: void DMSendIMA()
4702: {
4703:     SDMDMLLeader leader;
4704:     leader.source.setAddress(Address_DM);
4705:     leader.source.setSensorID(1);
4706:     leader.source.setPort(PORT_DM_ELECTION);
4707:     leader.SendTo(NetworkManager); //Send a pseudo IMA message to the Network Manager
4708:     printf("Sending IMA Msg to the Network Manager... \n");
4709: }
4710:
4711:
4712:
//*****
****
4713: //Description: Waits until the timeOut expires and then checks the Data Manager address
4714: // selected by the Network Manager to see if it matches itself. If the addresses match
4715: // this instance was selected as leader, otherwise as a backup Data Manager.
4716: //Input: MessageManger* - A pointer to the message managaer used in the RunBackupListener
4717: // function.
4718: //Output: bool - true if selected as leader, false otherwise
4719:
//*****
****
4720: bool AmIPrimary(MessageManager* mm)
4721: {
4722:     SDMDMLLeader response;
4723:     char buf[BUFSIZE];
4724:     long type;
4725:     bool isPrimary = false;
4726:     bool resolved = false;
4727:     unsigned int timeOut = 5;
4728:
4729:     unsigned long addr;
4730:     sleep(timeOut);
4731:
4732:     debug_f(3, "Election Timer Expired... \n");
4733:     resolved = true;
4734: #ifdef BUILD_FOR_PNPSAT

```

```

4735: struct hostent *he;
4736: he=gethostbyname("datamanager.spacewire");
4737: memcpy(&addr, he->h_addr, sizeof(addr));
4738: #endif
4739: #ifdef PNP_FAKE
4740: mm->BlockGetMessage(buf);
4741: if(buf[0] == SDM_DMLeader)
4742: {
4743:     response.Unmarshal(buf);
4744:     addr = response.source.getAddress();
4745: }
4746: #endif
4747: DataManager.setAddress(GetNodeAddress(spacewire));
4748:     debug_f(3, "Leader DM Address: 0x%lx My Address: 0x%lx \n", addr,
DataManager.getAddress());
4749: if(addr == DataManager.getAddress())
4750: {
4751:     isPrimary = true;
4752:     debug_f(1, "Selected by NM as leader \n");
4753: }
4754: else
4755: {
4756:     isPrimary = false;
4757:     printf("Data Manager running as backup... \n");
4758:     DataManager.setAddress(addr);
4759:     DataManager.setPort(PORT_DM);
4760: }
4761: return isPrimary;
4762: }
4763:
4764:
4765:
//*****
****
4766: //Description: This function allows a backup data manager to alert the leader dm of its
4767: // location and contains the basic listen loop for a backup data manager. It listens
4768: // for xTEDS registrations and cancelations, subscription type requests as well as an
4769: // SDMDMLeader message from the Heartbeat thread to alert that the leader has failed.
4770: //Input: MessageManger* - A pointer to the message manager used in the RunBackupListener
4771: // function.
4772: //Output: None

```

```

4773:
//*****
****
4774: void NonElected(MessageManager* mm)
4775: {
4776:   char buf[LARGE_MSG_BUFSIZE];
4777:   SDMReady tempready;
4778:   SDMReady backupAlert;
4779:   SDMDMLLeader DMleader;
4780:   SDMElection election;
4781:   SDMComponent_ID SenderId;
4782:   SDMAck ack;
4783:   struct sockaddr_in s;
4784:   s.sin_port = 0;
4785:   s.sin_addr.s_addr = 0;
4786:
4787:   long length;
4788:   pthread_attr_t attr;
4789:   pthread_t xtedThread, CancelThread;
4790:   SDMComHandle ComHandle;
4791:
4792:   pthread_attr_init(&attr);
4793:   pthread_attr_setstacksize(&attr,DM_STACK_SIZE);
4794:   debug_f(1, "Running as backup.. \n");
4795:
4796:   sleep(1); //Make sure Leader DM listen thread is up and going
4797:   backupAlert.destination.setAddress(GetNodeAddress(spacewire));
4798:   backupAlert.destination.setPort(PORT_DM_ELECTION);
4799:   backupAlert.SendTo(DataManager);
4800:   debug_f(1, "Informing leader DM of my address: 0x%lx \n",
backupAlert.destination.getAddress());
4801:
4802:
4803:   bool electionNeeded = false;
4804:   while(!electionNeeded)
4805:   {
4806:     while(mm->IsReady() == true)
4807:     {
4808: #ifdef WIN32
4809:       mm->GetMsg(buf /* return */,length /* return */, ComHandle /* return */);
4810: #else
4811:       mm->GetMessage(buf /* return */,length /* return */, ComHandle /* return */);

```

```

4812: #endif
4813:      // WARNING: ComHandle must be "cleaned-up" by eventually issuing ComHandle-
4814:      >DoCleanup(),
4815:      // either explicitly or by destroying an xTEDSPParameters that uses it (only when TCP)
4816:      SenderId.setAddress(s.sin_addr.s_addr);
4817:      SenderId.setPort(ntohs(s.sin_port));
4818:      switch(buf[0])
4819:      {
4820:          case SDM_DMLeader:
4821:              DMleader.Unmarshal(buf);
4822:              debug_f(3,"Received SDMDMLeader message while running with running_flag
of %x \n",DMleader.running_flag);
4823:              if(DMleader.running_flag == 'e')
4824:              {
4825:                  electionNeeded = true;
4826:              }
4827:              break;
4828:          case SDM_Election:
4829:              break;
4830:          case SDM_Heartbeat:
4831:              QueueHeartbeat(buf, length);
4832:              break;
4833:          case SDM_Ready:
4834:              tempready.Unmarshal(buf);
4835:              if(tempready.destination.getPort() != 0)
4836:              {
4837:                  printf("Setting TM address to: 0x%lx port: %li \n",
tempready.destination.getAddress(), tempready.destination.getPort());
4838:                  TaskManager = tempready.destination;
4839:              }
4840:              break;
4841:          case SDM_xTEDS:
4842:              {
4843:                  SDMxTEDS temp;
4844:                  temp.Unmarshal(buf);
4845:
4846:                  xTEDSPParameters* pParam = new xTEDSPParameters(buf, length, SenderId,
ComHandle);
4847:
4848:                  // In backup mode, send the Ack immediately -- the registration thread doesn't need to

```

```

4849:                SendAckMessage(SDM_OK, false, COMPONENT_ID_INVALID, pParam-
>GetComHandle(), true);
4850:                // Cleanup the com handle because the below thread won't use it
4851:                pParam->ComCleanup();
4852:                //if(temp.source.getPort() != PORT_DM) //Backups shouldn't register the DM,
avoids getting copies
4853:                //{
4854:                    if (0 != pthread_create(&xtedThread, &attr, xTEDS, pParam))
4855:                        perror("Could not start the xTEDS registration thread. \n");
4856:                    pthread_detach(xtedThread);
4857:                //}
4858:            }
4859:            break;
4860:            case SDM_CancelxTEDS:
4861:            {
4862:                xTEDSParameters* pParam = new xTEDSParameters(buf, length, SenderId,
ComHandle);
4863:
4864:                // In backup mode, send the Ack immediately -- the cancellation thread doesn't
need to
4865:                SendAckMessage(SDM_OK, false, COMPONENT_ID_INVALID, pParam-
>GetComHandle(), true);
4866:                // Cleanup the com handle because the below thread won't use it
4867:                pParam->ComCleanup();
4868:
4869:                if (0 != pthread_create(&CancelThread, &attr, CancelxTEDS, pParam))
4870:                    perror("Could not start the cancel xTEDS thread. \n");
4871:                pthread_detach(CancelThread);
4872:                break;
4873:            }
4874:            case SDM_ReqReg:
4875:                BackupModifyReqRegSubscription(buf);
4876:                break;
4877:            case SDM_Search:
4878:                BackupModifySearchSubscription(buf);
4879:                break;
4880:            case SDM_VarReq:
4881:                BackupModifyVarReqSubscription(buf);
4882:                break;
4883:            case SDM_TaskError:
4884:                BackupTaskError(buf);
4885:                break;

```



```

4886:         case SDM_Consume:
4887:             BackupConsume(buf);
4888:             break;
4889:             default:
4890:                 break;
4891:         }
4892:     }
4893: }
4894: }
4895:
4896:
4897:
//*****
****
4898: //Description: This thread handles the heartbeats for Data Managers and backup DMs.
4899: // Sends an SDMDMLeader message to the backup listener if the leader fails.
4900: //Input: None
4901: //Output: None
4902:
//*****
****
4903: void* Heartbeat (void* arg)
4904: {
4905:     //printf("Starting the Heatbeat Thread....sleeping for 5 secs \n");
4906:     sleep(5);                                //Give the UDPListener thread a little time
4907:     int num_sent = 0;                        //counter for the number of messages sent
4908:     int num_received = 0;                    //counter for the number of messages received
4909:
4910:     struct pollfd dm_poll_fd;                //poll struct
4911:     dm_poll_fd.events = POLLIN | POLLPRI;
4912:     dm_poll_fd.fd = dm_heartbeat_pipe[0];
4913:
4914:     int poll_res = 0;
4915:     char buf[BUFSIZE];                        //buffer for receiving message
4916:     long length;                             //Length of the heartbeat message being read
4917:     bool electionNeeded = false;
4918:
4919:     while (!electionNeeded)
4920:     {
4921:         if (!g_bElectedDm) //Backup DM
4922:         {
4923:             bool mainIsAlive = true;

```

```

4924:     SDMHeartbeat pulse;
4925:
4926:     while (mainIsAlive)
4927:     {
4928:         // send heart beat
4929:         SDMComponent_ID tempDM;
4930:         tempDM = DataManager;
4931:         tempDM.setPort(PORT_DM);
4932:         debug_f(3, "Heartbeat Sent to DM at: %0x1x port: %li \n", tempDM.getAddress(),
tempDM.getPort());
4933:         pulse.source.setAddress(GetNodeAddress());
4934:         pulse.source.setPort(PORT_DM_ELECTION);
4935:         pulse.SendTo(tempDM);
4936:
4937:         // poll for response, with a 5 second timeout
4938:         poll_res = poll (&dm_poll_fd,1,5000);
4939:         if (poll_res < 0)
4940:         {
4941:             perror ("Heartbeat: Poll error. \n");
4942:         }
4943:         else if (poll_res == 0) // when poll returns a zero it means there was a time out
4944:         {
4945:             printf("Data Manager not responding!! Sending IMA message \n");
4946:             mainIsAlive = false;
4947:         }
4948:         else // clear the pipe by reading the message
4949:         {
4950:             // Read message length from the pipe
4951:             read(dm_heartbeat_pipe[0], &length, sizeof(long));
4952:
4953:             // Read message from pipe
4954:             if (length > 0 && length < BUFSIZE)
4955:             {
4956:                 read(dm_heartbeat_pipe[0], buf, length);
4957:                 debug_f(3, "Response heartbeat received from DM \n");
4958:             }
4959:         }
4960:
4961:         if(poll_res != 0) //Sleep only if no
4962:         {
4963:             sleep(5);    // only send heart beats every 5 seconds

```

```

4964:     }
4965: }//end while
4966: electionNeeded = true;
4967:
4968: //Send a message to the Backup listener thread to let it know an election is needed
4969: SDMComponent_ID backupDM;
4970: backupDM.setAddress(GetNodeAddress());
4971: backupDM.setPort(PORT_DM_ELECTION);
4972: SDMDMLeader electionNeededMsg;
4973: electionNeededMsg.running_flag = 'e';
4974: electionNeededMsg.SendTo(backupDM);
4975: }
4976: else //Leader DM
4977: {
4978:     while (1)
4979:     {
4980:         //respond to any heart beats we have received while sleeping
4981:         if ((poll_res=poll(&dm_poll_fd,1,1000)) < 0)
4982:         {
4983:             perror ("Heartbeat: Poll error. \n");
4984:             break;
4985:         }
4986:         else if (poll_res > 0)
4987:         {
4988:             debug_f(3, "Heartbeat received from a backup DM \n");
4989:             // Read message length from the pipe
4990:             read(dm_heartbeat_pipe[0], &length, sizeof(long));
4991:
4992:             // Read message from pipe
4993:             if (length > 0 && length < BUFSIZE)
4994:             {
4995:                 read(dm_heartbeat_pipe[0], buf, length);
4996:
4997:                 // send a reply
4998:
4999:                 SDMHeartbeat temp;
5000:                 temp.Unmarshal(buf);
5001:
5002:                 SDMComponent_ID backup;
5003:                 backup = temp.source;
5004:

```

```

5005:         temp.source = DataManager;
5006:         temp.SendTo(backup);
5007:         debug_f(3, "Sending heartbeat to backup at: 0x%lx port: %li \n", backup.getAddress(),
backup.getPort());
5008:     }
5009: }
5010: } //while
5011: } //else
5012: } //while
5013: return NULL;
5014: }
5015:
5016: void QueueHeartbeat(char buf[], long length)
5017: {
5018:     SDMHeartbeat temp;
5019:     temp.Unmarshal(buf);
5020:
5021:     //debug_f (3, "Queuing heartbeat for DM \n");
5022:     /* Write the heartbeat message into the heartbeat pipe for the heartbeat thread */
5023:     // First, write the length of the message
5024:     if (write(dm_heartbeat_pipe[1], &length, sizeof(long)) < 0)
5025:         printf("Could not write heartbeat length to pipe \n");
5026:     // Write the actual message
5027:     if (write(dm_heartbeat_pipe[1], buf, length) < 0)
5028:         printf("Could not write heartbeat msg to pipe \n");
5029: }
5030:
5031:
5032: void BackupModifyReqRegSubscription(const char* msgBuf)
5033: {
5034:     SDMReqReg msgRequest;
5035:
5036:     if (msgRequest.Unmarshal(msgBuf) < 0)
5037:     {
5038:         printf("Invalid SDMReqReg message received. \n");
5039:         return ;
5040:     }
5041:     MessageReceived(&msgRequest);
5042:
5043:     debug_f(1,"command: Request Registrations Subscription Entry -- reply: %d item_name: \"%s \"
quallist: \"%s \" \n", msgRequest.reply, msgRequest.item_name, msgRequest.quallist);

```

```

5044:
5045: // If the reply style isn't SDM_REQREG_CURRENT (i.e. a subscription needs to be entered or
cancelled)
5046: if(msgRequest.reply != SDM_REQREG_CURRENT)
5047: {
5048:     debug_f(2,"Backing up ReqReg Subscription... \n");
5049:     pthread_mutex_lock(&subscription_mutex);
5050:     subscribers.addOrRemoveSubscription(msgRequest.reply,          msgRequest.destination,
msgRequest.source,          msgRequest.device,          msgRequest.interface,          msgRequest.item_name,
msgRequest.quallist, msgRequest.id, REQ_REG_FUTURE, debug);
5051:     pthread_mutex_unlock(&subscription_mutex);
5052: }
5053: }
5054:
5055: void BackupModifySearchSubscription(const char* msgBuf)
5056: {
5057:     SDMSearch msgSearch;
5058:
5059:     if (msgSearch.Unmarshal(msgBuf) < 0)
5060:     {
5061:         printf("Invalid SDMSearch message received. \n");
5062:         return ;
5063:     }
5064:     MessageReceived(&msgSearch);
5065:
5066:     debug_f(1,"command: Search Subscription Entry -- reply: %d reg_expr:  \"%s \" \n",
msgSearch.reply, msgSearch.reg_expr);
5067:     //
5068:     // The reply is one of SDM_SEARCH_CURRENT_AND_FUTURE or
SDM_SEARCH_CANCEL for this to match
5069:     if(msgSearch.reply != SDM_SEARCH_CURRENT)
5070:     {
5071:         debug_f(2,"Backing up Search Subscription... \n");
5072:         pthread_mutex_lock(&subscription_mutex);
5073:         subscribers.addOrRemoveSubscription(msgSearch.reply,          msgSearch.destination,
msgSearch.source, NULL, NULL, msgSearch.reg_expr, NULL, msgSearch.id, SEARCH_REPLY,
debug);
5074:         pthread_mutex_unlock(&subscription_mutex);
5075:     }
5076: }
5077:
5078: void BackupModifyVarReqSubscription(const char* msgBuf)

```

```

5079: {
5080:   SDMVarReq msgVarRequest;
5081:
5082:   if (msgVarRequest.Unmarshal(msgBuf) < 0)
5083:   {
5084:     printf("Invalid SDMVarReq message. \n");
5085:     return ;
5086:   }
5087:   MessageReceived(&msgVarRequest);
5088:
5089:   debug_f(1,"command: Variable Request Subscription Entry -- ID: %d variable: \"%s\" \n",
msgVarRequest.id, msgVarRequest.variable);
5090:   //
5091:   // Add a subscription entry if entered
5092:   if(msgVarRequest.reply != SDM_VARREQ_CURRENT)
5093:   {
5094:     debug_f(2,"Backing up VarReqReg Subscription... \n");
5095:     char interface_id[8] = "";
5096:     if(msgVarRequest.msg_id.getInterfaceMessagePair() != 0 || msgVarRequest.reply !=
SDM_VARREQ_CANCEL)
5097:       sprintf(interface_id,"%d",msgVarRequest.msg_id.getInterfaceMessagePair());
5098:
5099:     pthread_mutex_lock(&subscription_mutex);
5100:     subscribers.addOrRemoveSubscription(msgVarRequest.reply, msgVarRequest.destination,
msgVarRequest.source, NULL, interface_id, msgVarRequest.variable, NULL, msgVarRequest.id,
VAR_REQ_REPLY, debug);
5101:     pthread_mutex_unlock(&subscription_mutex);
5102:   }
5103: }
5104:
5105:
5106: void BackupTaskError(const char* msgBuf)
5107: {
5108:   SDMTaskError msgError;
5109:   SDMComponent_ID RequesterId;
5110:   if (msgError.Unmarshal(msgBuf) < 0)
5111:   {
5112:     printf("Invalid SDMTaskError message. \n");
5113:     return ;
5114:   }
5115:   MessageReceived(&msgError);
5116:   RequesterId = msgError.source;

```

```

5117:  debug_f(1,"command: Task Error -- TaskName: \"%s\" \n", msgError.filename);
5118:
5119:  debug_f(2,"Backing up SDMTaskError Message... \n");
5120:
5121:
5122: // xTEDSLibraryListNode* tempNode = xTEDSList.head;
5123: // int count = 0;
5124: // printf("***xTEDS Library List - size: %i*** \n", xTEDSList.size());
5125: // while(tempNode!=NULL)
5126: // {
5127: //     printf("xTEDSList[%i] PID: %li Error PID: %li Add: 0x%lx Error Add: 0x%lx Active: %i
5128: //         count, tempNode->data->getPid(), msgError.pid, tempNode->data->getAddress(),
5129: //         RequesterId.getAddress(), tempNode->data->getActive());
5130: //     tempNode = tempNode->next;
5131: //     count++;
5132: // }
5133: // printf("***** \n");
5134: //
5135: // Start looking for the xTEDS to cancel
5136: xTEDSLibraryListNode* node = xTEDSList.head;
5137: int iRemoveRef = 0;
5138: while(node!=NULL)
5139: {
5140:     node->data->inUse->Wait();
5141:     if(node->data->getAvailable() == false)
5142:     {
5143:         if (RequesterId.getAddress() == node->data->getAddress())
5144:         {
5145:             if(node->data->getActive() == true)
5146:             {
5147:                 if (node->data->getPid() == msgError.pid)
5148:                 {
5149:                     debug_f(2, "Sensor name to cancel is %s \n", node->data->getName());
5150:                     node->data->inUse->Signal();
5151:                     break;
5152:                 }
5153:             }
5154:         }
5155:     }

```

```

5156:     iRemoveRef++;
5157:     node->data->inUse->Signal();
5158:     node = node->next;
5159: }
5160: addLibrary.Wait();
5161:
5162: //
5163: // If the xTEDS couldn't be found, report error and exit
5164: if(node==NULL)
5165: {
5166:     debug_f(1,"Could not identify the sensor to cancel. \n");
5167:     addLibrary.Signal();
5168:     return ;
5169: }
5170:
5171: node->data->inUse->Wait();
5172: node->data->setActive(false);
5173: // Keep this task posted so a ReqReg won't re-post it -- the PM will
5174: // immediately restart it
5175: node->data->setPosted(true);
5176: const unsigned long CancelSensorID = node->data->getSensorID();
5177: const SDMComponent_ID& CancelComponentId = node->data->getComponentID();
5178: node->data->inUse->Signal();
5179: //
5180: // Notify any devices about the cancelled application (ala SDMDeltesub)
5181: pthread_mutex_lock(&sensor_subs_mutex);
5182: g_SensorSubscriptions.DeleteAll(CancelComponentId, false);
5183: g_SensorSubscriptions.ProviderFinish(CancelComponentId);
5184: pthread_mutex_unlock(&sensor_subs_mutex);
5185: //
5186: //Remove any subscriptions that the failed task had
5187: pthread_mutex_lock(&subscription_mutex);
5188: subscribers.removeReqRegSubscription(SDM_REQREG_CANCEL,      CancelComponentId,
debug);
5189: pthread_mutex_unlock(&subscription_mutex);
5190:
5191: PrintxTEDS();
5192:
5193: addLibrary.Signal();
5194:
5195: }

```



```

5196:
5197: void BackupConsume(const char* msgBuf)
5198: {
5199:     SDMConsume msgConsumeRequest;
5200:     SDMSubreqst msgSubRequest;
5201:     xTEDSLibraryListNode* node;
5202:
5203:     if(msgConsumeRequest.Unmarshal(msgBuf) < 0)
5204:     {
5205:         printf("Invalid SDMConsume message \n \n");
5206:         return;
5207:     }
5208:     MessageReceived(&msgConsumeRequest);
5209:
5210:     debug_f(1, "command: Consume -- port: %hu sensorID: %ld msgID: 0x%x\n",msgConsumeRequest.destination.getPort(),msgConsumeRequest.source.getSensorID(),msgConsumeRequest.msg_id.getInterfaceMessagePair());
5211:     //
5212:     // Get the node structure for the sensor requested
5213:
5214:
5215:
5216:
5217: // int count = 0;
5218: // for(xTEDSLibraryListNode* tempNode = xTEDSList.head; tempNode != NULL; tempNode = tempNode->next)
5219: // {
5220: //     printf("Node[%i] SID: %ld \n", count, tempNode->data->getSensorID());
5221: //     count++;
5222: // }
5223:
5224:
5225:
5226: node = MatchSID(msgConsumeRequest.source.getSensorID());
5227: if(node==NULL) //If node is NULL reaches xsize there is no matching sensorID and request is invalid
5228: {
5229:     printf("Could not find the sensor requested! \n");
5230:     return;
5231: }
5232: node->data->inUse->Signal();
5233: //

```

```

5234: // Put message together for the requested sensor
5235: msgSubRequest.destination = msgConsumeRequest.destination;
5236: msgSubRequest.source = msgConsumeRequest.source;
5237: msgSubRequest.msg_id = msgConsumeRequest.msg_id;
5238: //
5239: // If this is a request for the Data Manager, add a subscription table entry
5240: if(node->data->getSensorID() == DM_SENSOR_ID)
5241: {
5242:     node->data->inUse->Signal();
5243:     SDMComponent_ID SubscriberId;
5244:     SubscriberId.setAddress(msgSubRequest.destination.getAddress());
5245:     SubscriberId.setPort(msgSubRequest.destination.getPort());
5246:     // Add a subscription entry
5247:     Subscribe(SubscriberId, msgSubRequest.msg_id);
5248:     debug_f(1," \n");
5249:     return;
5250: }
5251: // Get the fault ID for the requested message
5252: msgSubRequest.fault_id = node->data->xtedsTree-
>getNotificationFaultMsgID(msgConsumeRequest.msg_id);
5253:
5254: //
5255: // Increment the number of connections to the xTEDS
5256: int n = node->data->getConnections();
5257: n++;
5258: node->data->setConnections(n);
5259: node->data->inUse->Signal();
5260: //
5261: // If the sensor ID is filled in for the subscriber, it has an xTEDS and should be marked in
5262: // the DM's provider subscription list so the DM can send out cancellations upon any type of
failure
5263: if (msgSubRequest.destination.getSensorID() != 0 && msgSubRequest.destination.getSensorID()
!= 1)
5264: {
5265:     debug_f(3, "Adding subscription to the g_SensorSubscriptions list... \n");
5266:     pthread_mutex_lock(&sensor_subs_mutex);
5267:     g_SensorSubscriptions.Add(msgSubRequest.source /*provider*/,
5268:         msgSubRequest.destination/*subscriber*/, msgSubRequest.msg_id);
5269:     pthread_mutex_unlock(&sensor_subs_mutex);
5270: }
5271: }
5272:

```

```
5273: //////////////////////////////////////
5274: //
5275: //      End Data Manager backup functions.
5276: //
5277: //////////////////////////////////////
5278: #endif // #ifdef PNP_BACKUP
```

File: sdm/dm/xTEDSSegmentBuilder.cpp

```
1: #include <stdlib.h>
2: #include <string.h>
3: #include <stdio.h>
4: #include <unistd.h>
5: #include "xTEDSSegmentBuilder.h"
6:
7: #define XTEDS_EMPTY    0
8: #define XTEDS_RECEIVED  1
9:
10: /*
11:  * Default constructor; initially the linked list of xTEDSSegmentNodes is empty.
12:  * The list is built as requests are made.
13:  *
14:  */
15: xTEDSSegmentBuilder::xTEDSSegmentBuilder()
16: {
17: }
18:
19: /*
20:  * Destructor; this method is mainly responsible for freeing the xTEDSSegmentNodes list.
21:  *
22:  *
23:  */
24: xTEDSSegmentBuilder::~~xTEDSSegmentBuilder()
25: {
26: }
27:
28: /*
29:  * ApplySegment is responsible for copying in the xTEDS of the segment number of the passed
30:  * Message. If there is no xTEDSSegmentNode corresponding to the xTEDS provider, one is
31:  * created. In other words, there is no other function to add a xTEDSSegmentNode, this one
32:  * is to be used.
33:  * Params:
34:  *     Message - The xTEDS to apply.
35:  * Returns:
36:  *     bool - True if the xTEDS was applied successfully, false otherwise.
37:  */
38: bool xTEDSSegmentBuilder::ApplySegment(const SDMxTEDS &Message)
39: {
```

```

40: //Be sure this is a segment first
41: if (!IsSegmentedxTEDS(Message))
42: {
43:     printf("xTEDSSegmentBuilder::ApplySegment - xTEDS not segmented. \n");
44:     return false;
45: }
46:
47: //Get the sequence number and the total segments from the xTEDS
48: unsigned char SequenceNumber = Message.xTEDS[0]; //Zero-based sequence number
49: unsigned char TotalSegments = Message.xTEDS[1]; //Total number of segments
50:
51: //If the sequence number is bigger than the advertised number of segments, or out of range
52: if (SequenceNumber+1 > TotalSegments || SequenceNumber >
MAX_XTEDS_SEQUENCE_VALUE || TotalSegments > MAX_XTEDS_SEQUENCE_VALUE)
53: {
54:     printf("xTEDSSegmentBuilder::ApplySegment - Sequence number out of range. \n");
55:     return false;
56: }
57:
58: //Get a reference to the xTEDS node
59: xTEDSSegmentNode *SegmentNode = FindNodeEntry(Message.source);
60:
61: //If the node doesn't exist, create it
62: if (SegmentNode == NULL)
63: {
64:     //This should be the first sequence number
65:     if (SequenceNumber != 0)
66:     {
67:         printf("xTEDSSegmentBuilder::ApplySegment - Sequence number not zero for new xTEDS.
\n");
68:         return false;
69:     }
70:     //Allocate the node
71:     SegmentNode = AddSegmentNode();
72:     //If malloc error, return error
73:     if (SegmentNode == NULL)
74:     {
75:         printf("xTEDSSegmentBuilder::ApplySegment - Segment node could not be obtained. \n");
76:         return false;
77:     }
78:     //Save the number of segments

```

```

79:     SegmentNode->NumSegments = static_cast<unsigned int>(TotalSegments);
80:     //Save the component ID of the xTEDS sender
81:     SegmentNode->xTEDSID = Message.source;
82:     //Allocate the buffer to use for the xTEDS, based on the number of segments
83:     //SegmentNode->xTEDSBuffer          =          new          char[(TotalSegments          *
SEGMENT_MAX_XTEDS_SIZE)];
84:     if          (sizeof(SegmentNode->xTEDSBuffer)          <          static_cast<unsigned
int>(TotalSegments*SEGMENT_MAX_XTEDS_SIZE))
85:     {
86:         printf("xTEDSSegmentBuilder:: xTEDS buffer not large enough for incoming segmented
xTEDS, not accepting. \n");
87:         DeleteNode(Message.source);
88:         return false;
89:     }
90:     //Clear the buffer
91:     memset(SegmentNode->xTEDSBuffer, 0, sizeof(SegmentNode->xTEDSBuffer));
92: }
93: //Node exists
94: else
95: {
96:     //If this is a timeout response, but the segment was received, no need to reapply, return true
97:     if(AlreadyReceived(SequenceNumber, SegmentNode))
98:         return true;
99:     //Check that this segment number is sent in order
100:    if (!SentInOrder(SequenceNumber, SegmentNode))
101:    {
102:        printf("xTEDSSegmentBuilder::ApplySegment - Segments out of order. \n");
103:        return false;
104:    }
105: }
106: //At this point, apply the segment at the end of the current xTEDS document (this assumes in
order message reception)
107: strncpy((SegmentNode->xTEDSBuffer          +          strlen(SegmentNode->xTEDSBuffer)),
Message.xTEDS+2, strlen(Message.xTEDS+2));
108: //Set this segment as having been received
109: SegmentNode->SegmentsReceived[SequenceNumber] = XTEDS_RECEIVED;
110: //Return success
111: return true;
112: }
113:
114: /*
115: * Checks to see if the segmented xTEDS is full and finished.

```

```

116: * Params:
117: *     Message - The xTEDS with the corresponding provider to check
118: * Returns:
119: *     bool - True if the xTEDS is finished, false if it is not finished.
120: */
121: bool xTEDSSegmentBuilder::CheckIsFinished(const SDMxTEDS &Message)
122: {
123:     //Get a pointer to the matching node
124:     xTEDSSegmentNode *SegmentNode = FindNodeEntry(Message.source);
125:     //If no matching entry, return that it is not finished
126:     if (SegmentNode == NULL)
127:         return false;
128:     //Otherwise, check to see if the full xTEDS is built
129:     for (unsigned int i = 0; i < SegmentNode->NumSegments; i++)
130:     {
131:         //If there is a segment not received, return false
132:         if (SegmentNode->SegmentsReceived[i] == XTEDS_EMPTY)
133:             return false;
134:     }
135:     //Otherwise all segments have been received, return true
136:     return true;
137: }
138:
139: /*
140: * Returns the full xTEDS for the corresponding xTEDS provider in Message. A successful call to
141: * this method also frees the xTEDSSegmentNode.
142: * Params:
143: *     Message - The provider with which to match the xTEDS
144: *     xTEDSOut [OUTPUT] - Pointer to the buffer to store the xTEDS, this must be at least
145: *     XTEDS_MAX_SIZE
146: *     MaxSize - The size of xTEDSOut
147: * Returns:
148: *     bool - True upon success, false if some error occurred
149: */
150: bool xTEDSSegmentBuilder::GetFullxTEDS(const SDMxTEDS &Message, char *xTEDSOut, int
MaxSize)
151: {
152:     //First, be sure that the xTEDS is fully built, and that xTEDSOut is not NULL
153:     if (!CheckIsFinished(Message) || xTEDSOut == NULL)
154:         return false;

```

```

155: //Now we can return the xTEDS
156: xTEDSSegmentNode *SegmentNode = FindNodeEntry(Message.source);
157: //To be safe, see that we have the node (this should have been handled above)
158: if (SegmentNode == NULL)
159:     return false;
160:
161: //Check to see that the DM's xTEDS buffer will be big enough
162: if (strlen(SegmentNode->xTEDSBuffer) > static_cast<unsigned int>(MaxSize))
163: {
164:     printf("Data Manager not built to support xTEDS of size %d (only up to %d) bytes.
165: \n",strlen(SegmentNode->xTEDSBuffer),MaxSize);
166:     DeleteNode(Message.source);
167:     return false;
168: }
169: //Copy the xTEDS
170: else
171:     strcpy(xTEDSOut, SegmentNode->xTEDSBuffer);
172:
173: //At this point, the xTEDS has been received, release the node
174: DeleteNode(Message.source);
175:
176: return true;
177: }
178: /*
179: * Determines whether the Message is of the segmented form. This is determined by checking the
180: * first two bytes of the xTEDS buffer within the SDMxTEDS message. If these two bytes are non-
181: * printable (from 0<=BYTE<=15), then this is a segment of an xTEDS. Otherwise it is a regular
182: * xTEDS registration.
183: * Params:
184: *     Message - the xTEDS message to check.
185: * Returns:
186: *     bool - True if the xTEDS is segmented, false otherwise.
187: *
188: */
189: bool xTEDSSegmentBuilder::IsSegmentedxTEDS(const SDMxTEDS &Message)
190: {
191:     unsigned char SeqNum = Message.xTEDS[0]; //First byte of the xTEDS buffer
192:     unsigned char NumSegments = Message.xTEDS[1]; //Second byte of the xTEDS buffer
193:
194:     //Check to see if this is a sequence number and a positive number of segments

```



```

195:  if (SeqNum  <=  MAX_XTEDS_SEQUENCE_VALUE  &&  NumSegments  <=
MAX_XTEDS_SEQUENCE_VALUE && NumSegments > 0)
196:      return true;
197:  return false;
198: }
199: /*
200: * Appends an xTEDSSegmentNode to the linked list, and returns a pointer to the node.
201: * Returns:
202: *      xTEDSSegmentNode* - Pointer to the added node, or NULL if there was an error
203: */
204: xTEDSSegmentNode* xTEDSSegmentBuilder::AddSegmentNode()
205: {
206:     //
207:     // Search the list to see if any segment nodes are available
208:     int NodeIndex = 0;
209:     for ( ; NodeIndex < MAX_SEGMENT_NODES; NodeIndex++)
210:     {
211:         if (NodeList[NodeIndex].IsInUse)
212:             continue;
213:         else
214:             break;
215:     }
216:     //
217:     // If no more entries
218:     if (NodeIndex == MAX_SEGMENT_NODES)
219:         return NULL;
220:     //
221:     // Clear out anything remaining
222:     memset(NodeList[NodeIndex].xTEDSBuffer, 0, sizeof (NodeList[NodeIndex].xTEDSBuffer));
223:     NodeList[NodeIndex].NumSegments = 0;
224:     for (unsigned int i = 0; i < sizeof(NodeList[NodeIndex].SegmentsReceived); i++)
225:         NodeList[NodeIndex].SegmentsReceived[i] = XTEDS_EMPTY;
226:     NodeList[NodeIndex].xTEDSID.setSensorID(0);
227:     NodeList[NodeIndex].xTEDSID.setAddress(0);
228:     NodeList[NodeIndex].xTEDSID.setPort(0);
229:     //
230:     // Set node active
231:     NodeList[NodeIndex].IsInUse = true;
232:
233:     return &NodeList[NodeIndex];
234: }

```

```

235: /*
236: * Finds the node in the list corresponding to the component id.
237: * Params:
238: *     xTEDSID - The component id to match the node.
239: * Returns:
240: *     xTEDSSegmentNode* - A pointer to the node found, or NULL if not found.
241: */
242: xTEDSSegmentNode* xTEDSSegmentBuilder::FindNodeEntry(const SDMComponent_ID
&xTEDSID)
243: {
244:     //
245:     // Traverse the list and find the node
246:     for (int NodeIndex = 0; NodeIndex < MAX_NUMBER_SEGMENTS; NodeIndex++)
247:     {
248:         if (NodeList[NodeIndex].IsInUse && NodeList[NodeIndex].xTEDSID == xTEDSID)
249:             return &NodeList[NodeIndex];
250:     }
251:     //
252:     // If the node was not found
253:     return NULL;
254: }
255: /*
256: * Deletes the xTEDSBuilderNode corresponding to the xTEDSID.
257: * Params:
258: *     xTEDSID - The identifier corresponding to the xTEDS to delete
259: * Returns:
260: *     bool - True if the node was found and deleted, false if an error occurred
261: */
262: bool xTEDSSegmentBuilder::DeleteNode(const SDMComponent_ID &xTEDSID)
263: {
264:     //
265:     // Find the node
266:     xTEDSSegmentNode *NodeToDelete = FindNodeEntry(xTEDSID);
267:
268:     if (NodeToDelete == NULL) return false;
269:     //
270:     // Set the node as inactive
271:     NodeToDelete->IsInUse = false;
272:     return true;
273: }
274: /*

```

```

275: * Determines whether an xTEDS segment was sent in order. This is necessary to make sure that
the xTEDS document
276: * is written such that each segments goes into the document as advertised in the segment numbers.
277: * Params:
278: *     SequenceNumber - The sequence number to check
279: *     CurrNode - The receiver node to check against
280: * Returns:
281: *     bool - True if this packet is in order, false if it is out of order
282: */
283: bool xTEDSSegmentBuilder::SentInOrder(unsigned char SequenceNumber, xTEDSSegmentNode
*CurrNode)
284: {
285:     //Traverse the flag list
286:     for (unsigned int i = 0; i < sizeof(CurrNode->SegmentsReceived); i++)
287:     {
288:         //If there are empty slots before this current sequence number, this was not sent in order
289:         if (i < SequenceNumber && CurrNode->SegmentsReceived[i] == XTEDS_EMPTY)
290:             return false;
291:         //If there are full slots at or after this sequence number, this was not sent in order
292:         if (i >= SequenceNumber && CurrNode->SegmentsReceived[i] == XTEDS_RECEIVED)
293:             return false;
294:     }
295:     return true;
296: }
297: /*
298: * Determines whether an xTEDS segment was already received.
299: * Params:
300: *     SequenceNumber - The sequence number to check
301: *     CurrNode - The receiver node to check against
302: * Returns:
303: *     bool - True if this packet has already been received, false otherwise
304: */
305: bool xTEDSSegmentBuilder::AlreadyReceived(unsigned char SequenceNumber,
xTEDSSegmentNode *CurrNode)
306: {
307:     //Be sure the index is not out of bounds
308:     if (SequenceNumber >= sizeof(CurrNode->SegmentsReceived))
309:         return false;
310:     //Check if we have received this segment
311:     return (CurrNode->SegmentsReceived[SequenceNumber] == XTEDS_RECEIVED);
312: }

```

File: sdm/dm/backupDMList.cpp

```
1: #include "backupDMList.h"
2:
3: #include <stdlib.h>
4: #include <string.h>
5:
6: DMBBackupList::DMBackupList() : m_pHead(NULL), m_pLastAccess(NULL)
7: {}
8:
9:
10: DMBBackupList::~DMBackupList()
11: {
12: DeleteList();
13: }
14:
15: bool DMBBackupList::AddNode(const SDMComponent_ID& NewDmId)
16: {
17: DMBBackupListNode* pNewNode = new DMBBackupListNode;
18: if (pNewNode == NULL)
19: {
20:     return false;
21: }
22:
23: pNewNode->m_idAddress = NewDmId;
24:
25: // If the list is empty, set the new node to the head
26: if (m_pHead == NULL)
27: {
28:     m_pHead = pNewNode;
29: }
30: // Otherwise, find the tail, then add
31: else
32: {
33:     DMBBackupListNode* pCur = m_pHead;
34:     for (; pCur->m_pNext != NULL; pCur = pCur->m_pNext)
35:         ;
36:     pCur->m_pNext = pNewNode;
37: }
38:
39: // Set the last access pointer
```

```

40: m_pLastAccess = pNewNode;
41: #ifdef DEBUG_DM_BACKUP_LIST
42: PrintList();
43: #endif
44: }
45:
46: void DMBBackupList::DeleteList()
47: {
48: DMBBackupListNode* pTemp = NULL;
49: for (DMBackupListNode* pCur = m_pHead; pCur != NULL; )
50: {
51:     pTemp = pCur->m_pNext;
52:     delete pCur;
53:     pCur = pTemp;
54: }
55: m_pHead = NULL;
56: }
57:
58: // Returns whether a node was added
59: bool DMBBackupList::AddIfNew(const SDMComponent_ID& DmId)
60: {
61: if (ListContains(DmId))
62:     return false;
63: else
64: {
65:     AddNode(DmId);
66:     return true;
67: }
68: }
69:
70: // Returns whether the list contains the given Id
71: bool DMBBackupList::ListContains(const SDMComponent_ID& Id)
72: {
73: DMBBackupListNode* pCur = Find(Id);
74:
75: // If pCur is not null, the list contains the node
76: return (pCur != NULL);
77: }
78:
79:
80: void DMBBackupList::SendMessageToAll(SDMmessage& Message)

```

```

81: {
82: for (DMBackupListNode* pCur = m_pHead; pCur != NULL; pCur = pCur->m_pNext)
83: {
84:     Message.SendTo(pCur->m_idAddress);
85: }
86: }
87:
88: // Returns whether the Id was found
89: bool DMBackupList::SendMessageTo(const SDMComponent_ID& Id, SDMmessage& Message)
90: {
91: DMBackupListNode* pCur = Find(Id);
92:
93: if (pCur == NULL)
94:     return false;
95:
96: Message.SendTo(Id);
97: return true;
98: }
99:
100: DMBackupListNode* DMBackupList::Find(const SDMComponent_ID& Id)
101: {
102:     // For efficiency
103:     if (m_pLastAccess != NULL && m_pLastAccess->m_idAddress == Id)
104:         return m_pLastAccess;
105:
106:     // Otherwise, do a linear search
107:     for (DMBackupListNode* pCur = m_pHead; pCur != NULL; pCur = pCur->m_pNext)
108:     {
109:         if (pCur->m_idAddress == Id)
110:         {
111:             m_pLastAccess = pCur;
112:             return pCur;
113:         }
114:     }
115:     return NULL;
116: }
117:
118: #ifdef DEBUG_DM_BACKUP_LIST
119: void DMBackupList::PrintList()
120: {
121:     printf("*****DM Backup List***** \n");

```

```
122:  char strId[128];
123:  for (DMBackupListNode* pCur = m_pHead; pCur != NULL; pCur = pCur->m_pNext)
124:  {
125:      pCur->m_idAddress.IDToString(strId, sizeof(strId));
126:      printf(" %s \n", strId);
127:  }
128:  printf("***** \n");
129: }
130: #endif
131:
132:
133:
```

File: sdm/dm/ProviderSubscription.cpp

```
1: #include "ProviderSubscription.h"
2:
3: ProviderSubscription::ProviderSubscription() : m_ProviderId(), m_SubscriberId(), m_MessageId()
4: {
5: }
6:
7: void ProviderSubscription::Set(const SDMComponent_ID& idProvider, const SDMComponent_ID&
idSubscriber, const SDMMessage_ID& idMessage)
8: {
9:   m_ProviderId = idProvider;
10:  m_SubscriberId = idSubscriber;
11:  m_MessageId = idMessage;
12: }
```


File: sdm/dm/xTEDSPParameters.cpp

```
1: #include "xTEDSPParameters.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <stdio.h>
6: #include <netinet/in.h>
7: extern "C"
8: {
9: #include "../common/MemoryUtils.h"
10: }
11: #ifdef WIN32
12: #include "unistd.h"
13: #endif
14:
15: xTEDSPParameters::xTEDSPParameters():m_MessageData(NULL),m_iDataSize(0),m_idSender(),m_com
Handle()
16: {
17: }
18:
19: xTEDSPParameters::xTEDSPParameters(const xTEDSPParameters&
b):m_MessageData(NULL),m_iDataSize(0),m_idSender(),m_comHandle()
20: {
21: m_MessageData = SDM_strndup(b.m_MessageData,b.m_iDataSize);
22: m_iDataSize = b.m_iDataSize;
23: m_idSender = b.m_idSender;
24: }
25:
26: xTEDSPParameters::xTEDSPParameters(char* buffer, int length, const SDMComponent_ID& Sender,
const SDMComHandle& ComHandle)
27: : m_MessageData(NULL),m_iDataSize(length),m_idSender(),m_comHandle(ComHandle)
28: {
29: m_MessageData = (char*)malloc(length + 1);
30: if(m_MessageData==NULL)
31: {
32:     printf("Error in mallocing space for the xTEDS or Update!!! \n");
33: }
34: else
35: {
36: memcpy(m_MessageData,buffer,length);
```

```

37: m_MessageData[length] = '\0';
38: }
39: m_idSender = Sender;
40: }
41:
42: xTEDSPParameters::~xTEDSPParameters()
43: {
44: if(m_MessageData!=NULL) free(m_MessageData);
45: m_comHandle.DoCleanup();
46: }
47:
48: xTEDSPParameters& xTEDSPParameters::operator=(const xTEDSPParameters& b)
49: {
50: if(m_MessageData!=NULL) free(m_MessageData);
51: m_MessageData = SDM_strndup(b.m_MessageData,b.m_iDataSize);
52: m_iDataSize = b.m_iDataSize;
53: m_idSender = b.m_idSender;
54: return *this;
55: }

```

File: sdm/dm/xTEDSLibraryList.cpp

```
1: #include "xTEDSLibraryList.h"
2: #include "xTEDSLibrary.h"
3: #include <stdlib.h>
4: #include <string.h>
5: extern "C"
6: {
7: #include "../common/MemoryUtils.h"
8: }
9:
10: struct xTEDSLibraryListNode* copyList(struct xTEDSLibraryListNode* list, struct
xTEDSLibraryListNode** tail)
11: {
12: struct xTEDSLibraryListNode* p;
13: struct xTEDSLibraryListNode* head;
14: head = (struct xTEDSLibraryListNode*)SDM_malloc(sizeof(struct xTEDSLibraryListNode));
15: p = head;
16: for(struct xTEDSLibraryListNode* cur=list; cur!=NULL; cur=cur->next)
17: {
18:     p->data = cur->data;
19:     if(cur->next!=NULL)
20:     {
21:         p->next = (struct xTEDSLibraryListNode*)SDM_malloc(sizeof(struct
xTEDSLibraryListNode));
22:     }
23:     else
24:     {
25:         p->next = NULL;
26:         *tail = p;
27:     }
28:     p = p->next;
29: }
30: return head;
31: }
32:
33: void deleteList(struct xTEDSLibraryListNode* p)
34: {
35: if(p==NULL) return;
36: deleteList(p->next);
37: if(p->data!=NULL)
```

```

38:     delete(p->data);
39: free(p);
40: }
41:
42: int xTEDSLibraryList::size()
43: {
44:     int size = 0;
45:     for(xTEDSLibraryListNode* p = head; p != NULL; p = p->next)
46:     {
47:         size++;
48:     }
49:     return size;
50: }
51:
52: xTEDSLibraryList::xTEDSLibraryList():head(NULL),tail(NULL)
53: {}
54:
55: xTEDSLibraryList::xTEDSLibraryList(const xTEDSLibraryList& b):head(NULL),tail(NULL)
56: {
57:     head = copyList(b.head,&tail);
58: }
59:
60: xTEDSLibraryList::~~xTEDSLibraryList()
61: {
62:     deleteList(head);
63: }
64:
65: void xTEDSLibraryList::addLibrary(xTEDSLibrary* data)
66: {
67:     struct xTEDSLibraryListNode* p = (struct xTEDSLibraryListNode*)SDM_malloc(sizeof(struct
xTEDSLibraryListNode));
68:     p->data = data;
69:     p->next = NULL;
70:     if(tail == NULL)
71:     {
72:         head = p;
73:         tail = p;
74:     }
75:     else
76:     {
77:         tail->next = p;

```

```

78:     tail = p;
79: }
80: }
81:
82: xTEDSLibraryList& xTEDSLibraryList::operator=(const xTEDSLibraryList& b)
83: {
84:     deleteList(head);
85:     head = copyList(b.head,&tail);
86:     return *this;
87: }
88:
89: // Get the device name for the sensor id specified in sdmCompId and store it in pBuffer.
90: bool xTEDSLibraryList::GetDeviceName(const SDMComponent_ID& sdmCompId, char* pBuffer,
unsigned int uiBufferSize)
91: {
92:     unsigned long ulSensorId = sdmCompId.getSensorID();
93:     bool bFound = false;
94:
95:     if (ulSensorId == 0 || ulSensorId == 1)
96:     {
97:         snprintf(pBuffer, uiBufferSize, "unknown");
98:         return false;
99:     }
100:
101:     xTEDSLibrary* pXteds = NULL;
102:     for (xTEDSLibraryListNode* pCur = head; pCur != NULL && !bFound ; pCur = pCur->next)
103:     {
104:         pXteds = pCur->data;
105:
106:         pXteds->inUse->Wait();
107:         if (!pXteds->getAvailable())
108:         {
109:             if (pXteds->getComponentID().getSensorID() == ulSensorId)
110:             {
111:                 snprintf(pBuffer, uiBufferSize, "%s", pXteds->getName());
112:                 bFound = true;
113:             }
114:         }
115:         pXteds->inUse->Signal();
116:     }
117:     if (!bFound)

```

```
118:         snprintf(pBuffer, uiBufferSize, "unknown");
119:
120:     return bFound;
121: }
```

File: sdm/dm/xTEDSLibrary.cpp

```
1: #include "xTEDSLibrary.h"
2:
3: #include <string.h>
4: #include <stdlib.h>
5: #include <sys/socket.h>
6: #include <sys/types.h>
7: #include <regex.h>
8: #include <netinet/in.h>
9: #include <arpa/inet.h>
10: #include "../common/Exception/SDMRegexException.h"
11: #include "../common/ErrorUtils.h"
12: extern "C"
13: {
14: #include "../common/MemoryUtils.h"
15: }
16:
17:
xTEDSLibrary::xTEDSLibrary():inUse(NULL),xtedsTree(NULL),available(false),connections(0),active(
false),pid(0),hub(0),posted(false),m_strSPANode(NULL),fullxTED(NULL),tempSPAHub(NULL),id(NU
LL),merged(false)
18: {
19: id = new SDMComponent_ID();
20: if (id == NULL)
21:     ErrorUtils::MemoryAllocError(__FUNCTION__);
22:
23: id->setSensorID(0);
24: id->setPort(0);
25: id->setAddress(0);
26: connections = 0;
27: pid = 0;
28: available = true;
29: active = false;
30: hub = 0;
31: posted = false;
32: inUse = new Sem(1);
33: merged = false;
34: }
35:
```

```

36:             xTEDSLibrary::xTEDSLibrary(const xTEDSLibrary&
b):inUse(NULL),xtedsTree(NULL),available(false),connections(0),active(false),pid(0),hub(0),posted(fals
e),m_strSPANode(NULL),fullxTED(NULL),tempSPAHub(NULL),id(NULL),merged(false)
37: {
38: if(fullxTED!=NULL) free(fullxTED);
39: fullxTED = SDM_strdup(b.fullxTED);
40: if(m_strSPANode!=NULL) free(m_strSPANode);
41: m_strSPANode = SDM_strdup(b.m_strSPANode);
42: if(xtedsTree!=NULL) delete(xtedsTree);
43: xtedsTree = b.xtedsTree;
44: if(id!=NULL) delete(id);
45: id = b.id;
46: if(tempSPAHub!=NULL) free(tempSPAHub);
47: tempSPAHub = b.tempSPAHub;
48: connections = b.connections;
49: pid = b.pid;
50: available = b.available;
51: active = b.active;
52: hub = b.hub;
53: posted = b.posted;
54: merged = b.merged;
55: }
56: /****
57: Destructor - Frees all dynamically allocated members.
58: ****/
59: xTEDSLibrary::~xTEDSLibrary()
60: {
61: if(xtedsTree!=NULL) delete(xtedsTree);
62: if(fullxTED!=NULL) free(fullxTED);
63: if(m_strSPANode!=NULL) free(m_strSPANode);
64: if(id!=NULL) delete(id);
65: if(tempSPAHub!=NULL) free(tempSPAHub);
66: delete(inUse);
67: }
68: /****
69: setxTEDS sets the xTEDS for this xTEDSLibrary. This function will also free any previously set
xTEDS and the associated
70: xtedsTree member. A new xtedsTree is built after parsing the xTEDS.
71: Params:
72:     new_xTEDS - The xTEDS to be set.
73: Returns:
74:     bool - True if the xTEDS was successfully parsed, false otherwise.

```



```

75: ****/
76: bool xTEDSLibrary::setxTEDS(char* new_xTEDS)
77: {
78: if(fullxTED!=NULL) free(fullxTED);
79: int length = strlen(new_xTEDS);
80: fullxTED = SDM_strndup(new_xTEDS,length);
81: if(xtedsTree!=NULL) delete(xtedsTree);
82: xtedsTree = new xTEDS();
83: if (xtedsTree == NULL)
84:     ErrorUtils::MemoryAllocError(__FUNCTION__);
85:
86: bool bParseResult = xtedsTree->Parse(new_xTEDS);
87: if (false == bParseResult)
88: {
89:     delete xtedsTree;
90:     xtedsTree = NULL;
91: }
92: return bParseResult;
93: }
94: /****
95: setAddress sets the IP address member of the "id" SDMComponent_ID member.
96: Params:
97:     new_ip - The character string IP address to set.
98: ****/
99: void xTEDSLibrary::setAddress(const SDMComponent_ID& new_idAddr)
100: {
101:     id->setAddress(new_idAddr.getAddress());
102: }
103: /****
104: setSensorID sets the sensorID member of the "id" SDMComponent_ID member.
105: ****/
106: void xTEDSLibrary::setSensorID(unsigned long new_sensorID)
107: {
108:     id->setSensorID(new_sensorID);
109: }
110: /****
111: setAvailable sets the "available" member, indicating whether this xTEDSLibrary node is being
used.
112: Params:
113:     new_available - The value to set, true if the xTEDSLibrary node is being used, false
otherwise.

```

```

114: *****/
115: void xTEDSLibrary::setAvailable(bool new_available)
116: {
117:     available = new_available;
118: }
119: /*****/
120:     setTargetPort sets the port member of the "id" SDMComponent_ID member.
121:     Params:
122:         new_targetport - The port value to set.
123: *****/
124: void xTEDSLibrary::setTargetPort(unsigned short new_targetport)
125: {
126:     id->setPort(new_targetport);
127: }
128: /*****/
129:     setConnections sets the number of connections to the xTEDS. This represents the number of
consumed
130:     data messages to the device or application.
131:     Params:
132:         new_connections - The new number of connections to the xTEDS.
133: *****/
134: void xTEDSLibrary::setConnections(int new_connections)
135: {
136:     connections = new_connections;
137: }
138: /*****/
139:     setActive sets whether the xTEDS of the application should be active in the SDM system. This is
mainly
140:     applicable to applications, the xTEDS is still registered but is not active for registrations until
141:     it is set to active again.
142:     Params:
143:         new_active - The value to indicate whether active
144: *****/
145: void xTEDSLibrary::setActive(bool new_active)
146: {
147:     active = new_active;
148: }
149: /*****/
150:     setPid sets the application PID number. This is the SDM process identifier number.
151:     Params:
152:         new_pid - The identifier number to set for this xTEDS.

```

```

153: *****/
154: void xTEDSLibrary::setPid(unsigned long new_pid)
155: {
156:     pid = new_pid;
157: }
158: /****
159:     setHub sets the hub number to which the device is connected. This is only applicable to devices,
and
160:     not applications.
161:     Params:
162:         new_hub - The new hub number value.
163: *****/
164: void xTEDSLibrary::setHub(int new_hub)
165: {
166:     hub = new_hub;
167: }
168: /****
169:     setSPANode sets the USB path of the connected device. This is only applicable to devices,
170:     and not applications.
171:     Params:
172:         new_locationAddress - The new USB path of the device.
173: *****/
174: void xTEDSLibrary::setSPANode(char* new_locationAddress)
175: {
176:     if(m_strSPANode!=NULL) free(m_strSPANode);
177:     m_strSPANode = SDM_strdup(new_locationAddress);
178: }
179: /****
180:     setSPAHub sets the SPAHub address of the connected device. This is only applicable to devices,
181:     and not applications.
182:     Params:
183:         new_spahub - The new SPAHub address to set.
184: *****/
185: void xTEDSLibrary::setSPAHub(char* new_spahub)
186: {
187:     if(tempSPAHub!=NULL) free(tempSPAHub);
188:     tempSPAHub = SDM_strdup(new_spahub);
189: }
190: /****
191:     setPosted sets whether the application has been posted to the TaskManager. This is applicable
192:     only to applications and not devices.

```

```

193: Params:
194:     new_posted - Whether the application has been posted.
195: *****/
196: void xTEDSLibrary::setPosted(bool new_posted)
197: {
198:     posted = new_posted;
199: }
200: /****
201:     setMerged set whether the xTEDS has been merged with information from the sdm.config file.
202: Params:
203:     new_merged - Whether the xTEDS has been merged
204: *****/
205: void xTEDSLibrary::setMerged(bool new_merged)
206: {
207:     merged = new_merged;
208: }
209: /****
210:     getName returns the name of the device from the associated xTEDS tree.
211: Returns:
212:     char * - The name of the device, or NULL if not set.
213: *****/
214: const char* xTEDSLibrary::getName(void)
215: {
216:     if (xtedsTree != NULL)
217:         return xtedsTree->getDeviceName();
218:     return NULL;
219: }
220: /****
221:     getxTEDS returns the xTEDS of the connected device or application.
222: Returns:
223:     char * - The xTEDS of the device, or NULL if the xTEDS hasn't been set.
224: *****/
225: const char* xTEDSLibrary::getxTEDS(void)
226: {
227:     return fullxTED;
228: }
229: /****
230:     getAddress returns the IP address, in number-dot-notation, of the connected device or application.
231: Returns:
232:     char * - The IP address of the device or application.
233: *****/

```

```

234: unsigned long xTEDSLibrary::getAddress(void)
235: {
236:     return id->getAddress();
237: }
238: /****
239:     getSensorID returns the sensor identifier number of the device or application.
240:     Returns:
241:         unsigned long - The sensor identifier number of the device or application.
242: *****/
243: unsigned long xTEDSLibrary::getSensorID(void)
244: {
245:     return id->getSensorID();
246: }
247: /****
248:     getAvailable returns whether the xTEDSLibrary slot is being used.
249:     Returns:
250:         bool - True if the xTEDSLibrary slot is in use, false otherwise.
251: *****/
252: bool xTEDSLibrary::getAvailable(void)
253: {
254:     return available;
255: }
256: /****
257:     getTargetPort returns the port of the device or application.
258:     Returns:
259:         long - The port number of the device or application.
260: *****/
261: unsigned short xTEDSLibrary::getTargetPort(void)
262: {
263:     return id->getPort();
264: }
265: /****
266:     getConnections returns the number of connections to this xTEDSLibrary (number of consumed
    messages).
267:     Returns:
268:         int - the number of connections to the xTEDSLibrary.
269: *****/
270: int xTEDSLibrary::getConnections(void)
271: {
272:     return connections;
273: }

```

```

274: /****
275:     getActive returns whether or not the this xTEDS is active.
276:     Returns:
277:         bool - Wether the xTEDS is active
278: *****/
279: bool xTEDSLibrary::getActive(void)
280: {
281:     return active;
282: }
283: /****
284:     getPid returns the SDM process identifier number of the application.
285:     Returns:
286:         unsigned long - The SDM process identifier number of the application.
287: *****/
288: unsigned long xTEDSLibrary::getPid(void)
289: {
290:     return pid;
291: }
292: /****
293:     getHub returns the hub number of the connected device.
294:     Returns:
295:         int - The hub number of the connected device.
296: *****/
297: int xTEDSLibrary::getHub(void)
298: {
299:     return hub;
300: }
301:
302: /****
303:     getLocationAddress returns the USB path of the connected device.
304:     Returns:
305:         char * - The USB path of the connected device, or NULL if not set.
306: *****/
307: /*NOW INLINE
308: char* xTEDSLibrary::getLocationAddress(void)
309: {
310:     return m_strSPANode;
311: }*/
312:
313: /****
314:     getSPAHub returns the SPA hub of the connected device.

```

```

315: Returns:
316:     char * - The SPA hub of the connected device, or NULL if not set.
317: *****/
318: const char* xTEDSLibrary::getSPAHub(void)
319: {
320:     if(xtedsTree!=NULL)
321:         return xtedsTree->getSPAUHub();
322:     else
323:         return tempSPAHub;
324: }
325: /****
326:     getSPAPort returns the SPA port of the connected device.
327: Returns:
328:     char * - The SPA port of the connected device, or NULL if not set.
329: *****/
330: const char* xTEDSLibrary::getSPAPort(void)
331: {
332:     if(xtedsTree!=NULL)
333:         return xtedsTree->getSPAUPort();
334:     else
335:         return NULL;
336: }
337: /****
338:     getPosted returns whether the application has been posted to the TaskManager.
339: Returns:
340:     bool - True if the application has been posted to the TaskManager, false otherwise.
341: *****/
342: bool xTEDSLibrary::getPosted(void)
343: {
344:     return posted;
345: }
346: /****
347:     setComponentID sets the component identifier for the application or device.
348: Params:
349:     new_id - The component identifier to set.
350: *****/
351: void xTEDSLibrary::setComponentID(SDMComponent_ID new_id)
352: {
353:     id->setAddress(new_id.getAddress());
354:     id->setSensorID(new_id.getSensorID());
355:     id->setPort(new_id.getPort());

```

```

356: }
357: /****
358:   GetComponentID returns the SDMComponent_ID of the device or application.
359:   Returns:
360:       SDMComponent_ID - The component identifier of the device or application
361: *****/
362: SDMComponent_ID xTEDSLibrary::GetComponentID(void)
363: {
364:   SDMComponent_ID temp;
365:   temp.setAddress(id->getAddress());
366:   temp.setSensorID(id->getSensorID());
367:   temp.setPort(id->getPort());
368:   return temp;
369: }
370: /****
371:   GetComponentKey returns the component key of the device or application from the xTEDS tree.
372:   Returns:
373:       char * - The component key of the device or application, or NULL if the xTEDS is not
374:       available.
375: *****/
376: const char* xTEDSLibrary::GetComponentKey(void)
377: {
378:   if (xtedsTree != NULL)
379:     return xtedsTree->GetComponentKey();
380:   return NULL;
381: }
382: /****
383:   getMerged returns whether the xTEDS has been merged with info from the sdm.config file
384:   Returns:
385:       bool - True if the xTEDS has been merged, false otherwise
386: *****/
387: bool xTEDSLibrary::getMerged(void)
388: {
389:   return merged;
390: }
391: /****
392:   operator= performs a shallow copy of the xTEDSLibrary object (the xtedsTree object is not deep-
393:   copied).
394: *****/
395: xTEDSLibrary& xTEDSLibrary::operator=(const xTEDSLibrary& b)
396: {

```



```

395:   if(fullxTED!=NULL) free(fullxTED);
396:   fullxTED = SDM_strdup(b.fullxTED);
397:   if(m_strSPANode!=NULL) free(m_strSPANode);
398:   m_strSPANode = SDM_strdup(b.m_strSPANode);
399:   if(xtedsTree!=NULL) delete(xtedsTree);
400:   xtedsTree = b.xtedsTree;
401:   if(id!=NULL) delete(id);
402:   id = b.id;
403:   if(tempSPAHub!=NULL) free(tempSPAHub);
404:   tempSPAHub = b.tempSPAHub;
405:   connections = b.connections;
406:   pid = b.pid;
407:   available = b.available;
408:   active = b.active;
409:   hub = b.hub;
410:   posted = b.posted;
411:   return *this;
412: }
413: /*
414:   Perform a regular expression search of the xTEDS based on Pattern.
415: */
416: RegexResult xTEDSLibrary::XtedsRegexSearchCapturesOnly(const RegularExpression& Pattern)
417: {
418:   return RegexSearchCapturesOnly(fullxTED, Pattern);
419: }

```

File: sdm/dm/Parse.cpp

```
1: //Parse.cpp
2:
3: #include <stdio.h>
4: #include <stdlib.h>
5: #include <string.h>
6: #include <unistd.h>
7: #include <fcntl.h>
8: #include "Parse.h"
9: #include "DM.h"
10:
11: ///////////////////////////////////////////////////////////////////
12: //
13: // ParseDeviceName takes the xted and finds the device name in the xted
14: //   buf - char pointer to a buffer containing the xted
15: //   sensor_buf - char pointer to a buffer to put the device name into
16: //   level - the level to print statements
17: //
18: //   returns whether this is a DEVICE or APPLICATION, -1 if not found
19: //
20: ///////////////////////////////////////////////////////////////////
21: int ParseDeviceName(char* strSource, char* strReturnName, unsigned int uiReturnNameSize, int
iDebug)
22: {
23: const char* STR_DEVICE = "Device";
24: const char* STR_APPLICATION = "Application";
25: size_t iTagStart = 0, iTagEnd = 0;
26: bool bFound = false, bCorrectTag = false;
27: char* curPos = NULL;
28: int prevPos = 0;
29: const size_t SOURCE_LENGTH = strlen(strSource);
30: bool bTryDevice = true;
31:
32: while (!bFound)
33: {
34:     if (bTryDevice) // Check for Device
35:         curPos = strstr(strSource + prevPos, STR_DEVICE);
36:     else // Check for Application
37:         curPos = strstr(strSource + prevPos, STR_APPLICATION);
38:
```

```

39:  if (curPos == NULL)
40:  {
41:      if (bTryDevice)
42:      {
43:          // Try application instead
44:          bTryDevice = false;
45:          prevPos = 0;
46:          continue;
47:      }
48:      else
49:          return -1;
50:  }
51:  else
52:  {
53:      prevPos = curPos - strSource + 1;
54:      // Find the opening < tag, if we encounter anything other than " ", we have not matched
55:      for (iTagStart = curPos - strSource - 1; iTagStart > 0; iTagStart--)
56:      {
57:          // Success
58:          if (strSource[iTagStart] == '<')
59:          {
60:              bCorrectTag = true;
61:              break;
62:          }
63:          // Match failure, try the next
64:          else if (strSource[iTagStart] != ' ')
65:          {
66:              bCorrectTag = false;
67:              break;
68:          }
69:      }
70:      if (!bCorrectTag)
71:          continue;
72:
73:      // Find the closing > tag, if we encounter any other "<", we have not matched
74:      for (iTagEnd = iTagStart + 1; iTagEnd < SOURCE_LENGTH; iTagEnd++)
75:      {
76:          // Success
77:          if (strSource[iTagEnd] == '>')
78:          {
79:              bCorrectTag = true;

```

```

80:         break;
81:     }
82:     // Match failure, try the next
83:     else if (strSource[iTagEnd] == '<')
84:     {
85:         bCorrectTag = false;
86:         break;
87:     }
88: }
89: if (!bCorrectTag)
90:     continue;
91:
92: if (GetAttribute (&strSource[iTagStart], "name", strReturnName, uiReturnNameSize,
iDebug) != 0)
93:     continue;
94:
95: // Attribute successfully found
96: if (bTryDevice)
97:     return DEVICE;
98: else
99:     return APPLICATION;
100: }
101: }
102: return -1;
103: }
104:
105: ///////////////////////////////////////////////////////////////////
106: //
107: // GetAttributes looks through the item_def to find the attribute listed
108: // strSource - a char* containing the message to look through
109: // strAttribute - a char* containing the attribute to look for
110: // strReturnValue - a char* array to store the attribute value of attribute
111: // uiAttributeSize - the size of the attribute_value array
112: // iDebug - the level to print at
113: //
114: // returns zero if the attribute was successfully found, -1 if not found or error
115: //
116: ///////////////////////////////////////////////////////////////////
117: int GetAttribute(const char* strSource, const char* strAttribute, char* strReturnValue, unsigned int
uiAttributeSize, int iDebug)
118: {

```

```

119: unsigned int i = 0, attrstart = 0, attrend = 1;
120: // flag == 1 means the attribute name has been found within the tag
121: // flag == 2 means the attribute value has been found
122: int flag = 0;
123: int numQuotes = 0;
124:
125: if (strSource == NULL || strAttribute == NULL || strReturnValue == NULL || uiAttributeSize ==
0)
126:     return -1;
127:
128: const size_t ATTR_LENGTH = strlen(strAttribute);
129: const size_t SOURCE_LENGTH = strlen(strSource);
130: for(i = 0; i < SOURCE_LENGTH; i++)
131: {
132:     //The number of quote marks must be even to assure we're matching an XML attribute and
not within a string literal
133:     if (strSource[i] == '"')
134:         numQuotes++;
135:     //Only match the attribute name if the number of quotes is even (can't be within string literal)
136:     if(strncmp(&strSource[i], strAttribute, ATTR_LENGTH) == 0 && numQuotes%2 == 0)
137:     {
138:         flag = 1;    //Found the start of the attribute name
139:         attrstart = i + ATTR_LENGTH;
140:     }
141:     else if (strSource[i] == '"' && flag == 1)
142:     {
143:         flag = 2;    //Found the start of the attribute value
144:         attrstart = i + 1;
145:     }
146:     else if (strSource[i] == '"' && i > attrstart && flag == 2) //Found close of attribute value
147:     {
148:         attrend = i;
149:         if (attrend < attrstart)    //Don't allow integer wrap if something went wrong
150:             return -1;
151:         if (attrend - attrstart > uiAttributeSize)
152:             return -1;
153:         else
154:         {
155:             strncpy(strReturnValue, &strSource[attrstart], attrend - attrstart);
156:             strReturnValue[attrend - attrstart] = '\0';
157:         }

```

```

158:         return 0;
159:     }
160:     // Don't pass this XML element, if so, return
161:     if (strSource[i] == '>')
162:         return -1;
163: }
164: return -1;
165: }
166: #ifdef BUILD_FOR_XTEDS_MERGING
167: /*
168:  Description:
169:      Merges the information gotten from the sdm.config file with the xTEDS
170:
171:  Input:
172:      xTED - the xTEDS to be merged
173:      config - the information from the sdm.config file
174:      level - the debug level to print messages on
175:
176:  Output:
177:      -1 - the merge did not succeed
178:      0 - the merge was successful
179:
180:  Changed:
181:      None
182:
183: */
184: int MergeConfigxTED(char *xTED, int xTEDBufLength, char *config, int level)
185: {
186:     int success = -1;
187:     char* device = "Device";
188:     int flag = 0;
189:     char* spahub = "spaUHub";
190:     int length = 0;
191:     int templength = 0;
192:     int j = 0;
193:     int offset = 0;
194:     int d = 0;
195:     int CurLength = 0, TempLength = 0;
196:
197:     for(int i = 0; i < (int)strlen(xTED); i++)
198:     {

```

```

199:    //Find the start of the DEVICE tag
200:    if(strncmp(&xTED[i],device,strlen(device)) == 0)
201:    {
202:        flag = 1;    //Start of device tag found
203:        if(level >= 4)
204:        {
205:            printf("Found the start of the Device tag \n");
206:        }
207:    }
208:    //Find an ending or the element spaUHub
209:    if((strncmp(&xTED[i],"/>",2) == 0 || strncmp(&xTED[i],">",1) == 0 ||
    strncmp(&xTED[i],spahub,strlen(spahub)) == 0) && flag == 1)
210:    {
211:        //Simple case <DEVICE ... />
212:        if(strncmp(&xTED[i],"/>",2) == 0)
213:            length = 2;
214:        //spaUHub case only for robohubs that are partially merged before runtime <DEVICE ...
    spaUHub="..."...>
215:        else if(strncmp(&xTED[i],spahub,strlen(spahub)) == 0)
216:        {
217:            templength = i;
218:            i += strlen(spahub)+2;
219:            while(xTED[i]!='>')
220:                i++;
221:            length = i - templength + 1;
222:            i = templength;
223:        }
224:        //<DEVICE ...>...</DEVICE>
225:        else
226:        {
227:            //Find start of next opening tag
228:            while(xTED[i+length]!='<')
229:                length++;
230:            //If this is a closing tag it will be </DEVICE>
231:            //This case for no tags between <Device..> and </Device>
232:            if(xTED[i+length+1]=='/')
233:            {
234:                while(xTED[i+length]!='>')
235:                    length++;
236:                length++;
237:            }

```

```

238:         else
239:         {
240:             //Find the end of the <DEVICE...> tag in the config info
241:             while(strncmp(&config[j],"/>",2) != 0 && strncmp(&config[j],">",1) != 0)
242:             {
243:                 j++;
244:             }
245:             //If DEVICE tag closes with /> then this is a simple merge
246:             if(strncmp(&config[j],"/>",2) == 0)
247:             {
248:                 offset = -2;
249:                 while(config[j-offset] != 0)
250:                 {
251:                     offset--;
252:                 }
253:                 length = 0;
254:             }
255:         else
256:         {
257:             //Find the start of the next tag in the config info
258:             while(config[j+offset] != '<')
259:                 offset++;
260:             if(config[j+offset+1] == '/')
261:             {
262:                 while(config[j+offset] != '>')
263:                     offset++;
264:                 offset++;
265:                 offset = 0 - offset;
266:             }
267:         else
268:         {
269:             CurLength = strlen(&xTED[i+1]);
270:             char *temp = (char*)malloc(CurLength+1);
271:             //copy everything after > to end of xTEDS into temp
272:             strcpy(temp,&xTED[i+1]);
273:             xTED[i] = ' ';
274:
275:             //copy config info up to end to DEVICE ending >
276:             if (CurLength + j+1 < xTEDBufLength)
277:             {
278:                 strncpy(&xTED[i+1],config,j+1);

```



```

279:             CurLength += j+2;
280:         }
281:
282:         i += j + 2;
283:         //Find any other elements that might be between the opening and closing
of DEVICE
284:         while(strncmp(&temp[d], "</Device>", 9) != 0)
285:             d++;
286:
287:         //Copy the elements found, most likely will just be QUALIFIER tags
288:         if (CurLength + d < xTEDBufLength)
289:         {
290:             strncpy(&xTED[i], temp, d);
291:             CurLength += d;
292:         }
293:         i += d;
294:         while(config[j] != '<')
295:             j++;
296:
297:         //copy the rest of the config info
298:         TempLength = strlen(&config[j]);
299:         if (CurLength + TempLength < xTEDBufLength)
300:         {
301:             strcpy(&xTED[i], &config[j]);
302:             CurLength += TempLength;
303:         }
304:
305:         while(strncmp(&xTED[i], "</Device>", 9) != 0)
306:             i++;
307:         i += 9;
308:
309:         //copy the last of the xTEDS back in
310:         strncpy(&xTED[i], &temp[d+9], xTEDBufLength - CurLength);
311:         free(temp);
312:         success = 0;
313:         if(level >= 3)
314:             printf("New merged xTEDS is: \n %s \n", xTED);
315:         return success;
316:     }
317: }
318: }

```

```

319:     }
320:     // Merge the simple case
321:     CurLength = strlen(&xTED[i+length]);
322:     char *temp = (char*)malloc(CurLength+1);
323:
324:     // Save the original xTEDS just after the <Device ... />
325:     strcpy(temp,&xTED[i+length]);
326:     xTED[i] = ' ';
327:
328:     // Merge in the config information
329:     TempLength = strlen(config);
330:     if (CurLength + TempLength < xTEDBufLength)
331:     {
332:         strcpy(&xTED[i+1],config);
333:         CurLength += TempLength;
334:     }
335:
336:     // Copy back in the previously saved xTEDS info
337:     strncpy(&xTED[i+strlen(config)+1+offset],temp, xTEDBufLength - CurLength);
338:     free(temp);
339:     success = 0;
340:     if(level >= 3)
341:         printf("New merged xTEDS is: \n %s \n",xTED);
342:     return success;
343: }
344: }
345: return success;
346:
347: }
348: #endif // #ifdef BUILD_FOR_XTEDS_MERGING
349: /*
350: Description:
351:     Find the hub path corresponding with the device and save the result in HubPathOut.
352:
353: Input:
354:     DevicePath - The path of the ASIM device whose robust hub to find
355:     HubPathOutLen - The size of HubPathOut
356: Output:
357:     HubPathOut - The buffer into which to fill the address of the hub
358:     return bool - True if found, false if not found or error
359: Changed:

```

```

360:     None
361: */
362: bool FindDevicesHubPath (const char *DevicePath, char *HubPathOut, unsigned int
HubPathOutLen, int DebugLevel)
363: {
364:     if (DevicePath == NULL || HubPathOut == NULL)
365:         return false;
366:
367:     char FileBuf[65536];
368:     char ReturnPath[128];
369:     int FileFd = open(SDM_CONFIG_FILE_NAME, O_RDONLY);
370:     int FileLength = -1;
371:
372:     if (FileFd < 0)
373:     {
374:         if (DebugLevel >= 2)
375:             printf("Could not open file %s to find corresponding hub.
\n",SDM_CONFIG_FILE_NAME);
376:         return false;
377:     }
378:
379:     // Read the file contents
380:     if ((FileLength = read(FileFd, FileBuf, sizeof(FileBuf)-2)) < 0)
381:     {
382:         close(FileFd);
383:         return false;
384:     }
385:     else
386:         close(FileFd);
387:     FileBuf[FileLength] = '\0';
388:
389:     // If there are multiple instances of DevicePath (there shouldn't be), try them all for a match
390:     bool IsPortMatched = false;
391:     char* NextStartingPosition = NULL;
392:     while (!IsPortMatched)
393:     {
394:         // Find the occurrence of the DevicePath in the config file
395:         char* DevicePathLocation = NULL;
396:         // If we have already found an instance of DevicePath in the config file, but not the correct
one,
397:         // try again at the last checked position
398:         if (NextStartingPosition == NULL)

```

```

399:         DevicePathLocation = strstr(FileBuf, DevicePath);
400:     else
401:         DevicePathLocation = strstr(NextStartingPosition, DevicePath);
402:
403:     if (DevicePathLocation == NULL)
404:     {
405:         if (DebugLevel >= 3)
406:             printf("Could not find hub path for spaUPort %s in configuration file.
407: \n",DevicePath);
408:         return false;
409:     }
410:     char* DeviceTag = DevicePathLocation;
411:
412:     // Get a pointer to the starting of this "<Device..." tag
413:     while (DeviceTag != FileBuf && strncmp(DeviceTag, "<Device", 7)!=0)
414:         DeviceTag++;
415:
416:     // Make sure the spaUPort value matches this DevicePath
417:     if (GetAttribute(DeviceTag, "spaUPort", ReturnPath, sizeof(ReturnPath), DebugLevel) == 0)
418:     {
419:         if (DebugLevel >= 3)
420:             printf("spaUPort value could not be retrieved for sensor path %s. \n",DevicePath);
421:         // See if there is another instance of DevicePath in the file
422:         NextStartingPosition = DevicePathLocation+1;
423:         continue;
424:     }
425:     if (strcmp(ReturnPath, DevicePath) != 0)
426:     {
427:         if (DebugLevel >= 3)
428:             printf("spaUPort value (%s) does not match sensor path (%s).",ReturnPath,
429: DevicePath);
430:         // See if there is another instance of DevicePath in the file
431:         NextStartingPosition = DevicePathLocation+1;
432:         continue;
433:     }
434:
435:     // Pull out the spaUHub
436:     if (GetAttribute(DeviceTag, "spaUHub", HubPathOut, HubPathOutLen, DebugLevel) == 0)
437:     {
438:         if (DebugLevel >= 3)
439:             printf("Could not find the SpaUHub value. \n");

```

```

438:          // If the spaUPort matched, but we couldn't get the spaUHub, return -- the spaUPort is a
unique value
439:          // in the config file, we can be sure that it can't be retrieved
440:          return false;
441:      }
442:      // This point is success
443:      IsPortMatched = true;
444:  }
445:  return true;
446: }
447:
448: /*
449:  Description:
450:      Find configuration information for an xTEDS from the sdm.config file
451:
452:  Input:
453:      type - the type of device this is
454:      info - a pointer to a memory location where the config info can be stored
455:      hub - the hub number
456:      port - the port number
457:      level - the level to print debug message on
458:
459:  Output:
460:      -1 - unable to find any config info for xTEDS
461:      0 - found config info
462:
463:  Changed:
464:      None
465:
466: */
467: int FindConfigInfo(char *type, char* info, char* hub, char* port, int level)
468: {
469:     int fd = 0;
470:     char file[65536];
471:     int value = 0;
472:     char *xTEDClose = "</xTEDS>";
473:     size_t start = 0;
474:     size_t end = -1;
475:     int found = 0;
476:     int usingPort = 0;
477:     int length = 0;

```

```

478:  int flag = 0;
479:  int portUsed = 1;
480:  int hubUsed = 1;
481:  char *spaHub = "SPA_U_hub";
482:  char *camelspaHub = "spaUHub";
483:  char *spaPort = "SPA_U_port";
484:  char *camelspaPort = "spaUPort";
485:  int started = 0;
486:
487:  memset(file,0,sizeof(file));
488:  if(strcmp(type,"Application")==0)
489:      return -1;
490:  if(strcmp(type, "Device") != 0)
491:      return -1;
492:  if(hub == NULL && strcmp(type,"Device") == 0)
493:  {
494:      if(level >= 3)
495:          printf("hub is NULL \n");
496:      return -1;
497:  }
498:  else if(port == NULL)
499:      found = 2;
500:  else if(port[0] == '\0')
501:      found = 2;
502:  else
503:  {
504:      found = 3;
505:      usingPort = 1;
506:  }
507:  if(level >= 4)
508:  {
509:      printf("Found is %d while looking for spahub: %s spaport: %s \n",found,hub,port);
510:  }
511:
512:  fd = open(SDM_CONFIG_FILE_NAME,O_RDONLY);
513:  if(fd == -1)
514:  {
515:      perror("Config file was unable to be opened! ");
516:      return -1;
517:  }
518:  value = read(fd,file,65535);

```

```

519:  if(level >= 2)
520:      printf("Read %d bytes from file %s \n",value,SDM_CONFIG_FILE_NAME);
521:  close(fd);
522:  if (value > 0)
523:      file[value] = '\0';
524:  else
525:      file[0] = '\0';
526:
527:  for(size_t i = 0; i < (int)strlen(file); i++)
528:  {
529:      if(strncmp(&file[i],type,strlen(type)) == 0 && started != 1)
530:      {
531:          start = i + strlen(type) + 1;
532:          flag = 1;
533:          started = 1;
534:      }
535:      if(strncmp(&file[i],spaHub,strlen(spaHub)) == 0 ||
536:      strncmp(&file[i],camelspaHub,strlen(camelspaHub)) == 0)
537:      {
538:          hubUsed = 0;
539:          if(strncmp(&file[i],spaHub,strlen(spaHub)) == 0)
540:              i += strlen(spaHub) + 2;
541:          else
542:              i += strlen(camelspaHub) + 2;
543:          if(level >= 4)
544:          {
545:              printf("Found SPA_U_hub qualifier with next char: %c%c%c%c%c flag: %d hubUsed:
546:              %d \n",file[i],file[i+1],file[i+2],file[i+3],flag,hubUsed);
547:          }
548:      }
549:      if(strncmp(&file[i],spaPort,strlen(spaPort)) == 0 ||
550:      strncmp(&file[i],camelspaPort,strlen(camelspaPort)) == 0)
551:      {
552:          portUsed = 0;
553:          if(strncmp(&file[i],spaPort,strlen(spaPort)) == 0)
554:              i += strlen(spaPort) + 2;
555:          else
556:              i += strlen(camelspaPort) + 2;
557:          if(usingPort == 0)
558:          {
559:              portUsed = 1;
560:              hubUsed = 1;

```

```

558:         found = 2;
559:         if(level >= 4)
560:         {
561:             printf("Found SPA_U_port qualifier, but current search is not using it, therefore
this is not a match. \n");
562:         }
563:     }
564:     else
565:     {
566:         if(level >= 4)
567:         {
568:             printf("Found SPA_U_port qualifier \n");
569:         }
570:     }
571: }
572: if(strncmp(&file[i],hub,strlen(hub)) == 0 && flag == 1 && hubUsed != 1)
573: {
574:     found--;
575:     hubUsed = 1;
576:     if(level >= 4)
577:     {
578:         printf("Found SPA_U_hub value \n");
579:     }
580: }
581: if(port != NULL)
582: {
583:     if(strncmp(&file[i],port,strlen(port)) == 0 && usingPort == 1 && flag == 1 &&
portUsed != 1)
584:     {
585:         found--;
586:         portUsed = 1;
587:         if(level >= 4)
588:         {
589:             printf("Found SPA_U_port value \n");
590:         }
591:     }
592: }
593: if(strncmp(&file[i],xTEDClose,strlen(xTEDClose)) == 0 && found == 1)
594: {
595:     end = i;
596: }

```



```

597:     if(strncmp(&file[i],"\"",1) == 0)
598:     {
599:         hubUsed = 1;
600:         portUsed = 1;
601:
602:     }
603:     if(strncmp(&file[i],xTEDClose,strlen(xTEDClose)) == 0 && found != 1)
604:     {
605:         if(port == NULL)
606:             found = 2;
607:         else if(port[0] == '\0')
608:             found = 2;
609:         else
610:         {
611:             found = 3;
612:             portUsed = 0;
613:         }
614:         hubUsed = 0;
615:         started = 0;
616:         if(level >= 4)
617:         {
618:             printf("Resetting values after end of tag found \n");
619:         }
620:     }
621:     if(strncmp(&file[i],">",1) == 0)
622:     {
623:         flag = 0;
624:     }
625:     if(end > start)
626:     {
627:         length = end - start;
628:         memcpy(info,&file[start],length);
629:         info[length] = '\0';
630:         return 0;
631:     }
632: }
633: return -1;
634: }
635: //////////////////////////////////////
636: //
637: // ParseItemName looks through the item name sent for any possible regex char and returns

```

```

638: // a value specifying whether or not any were found
639: // item - a char* that the item name will be stored in
640: // level - the debug level at which to print
641: //
642: // returns the REQTYPE_REGEX if any found otherwise 0
643: //
644: //////////////////////////////////////
645:
646: int ParseItemName(char *item, int level)
647: {
648:     int found = 0;
649:     const size_t ITEM_LENGTH = strlen(item);
650:     for(unsigned int i = 0; i < ITEM_LENGTH; i++)
651:     {
652:         if(item[i] == '*' || item[i] == '.' || item[i] == '[' || item[i] == ']' || item[i] == '{' || item[i] == '}' ||
item[i] == '(' || item[i] == ')' || item[i] == '\\' || item[i] == '+' || item[i] == '?' || item[i] == '|' || item[i] == '^'
|| item[i] == '$')
653:         {
654:             //A regular expressions character was found, return that type
655:             found = REQTYPE_REGEX;
656:             break;
657:         }
658:     }
659:     return found;
660: }
661:
662: /*
663:     Description:
664:         Get the spaHub value from the xTEDS
665:
666:     Input:
667:         buf - the xTEDS
668:         hub - a pointer to store the spaHub value in
669:         level - the level to print debug messages on
670:
671:     Output:
672:         -1 - unable to get the spaHub value
673:         0 - successfully got the spaHub value
674:
675:     Changed:
676:         None

```

```

677:
678: */
679: int GetSPAHub(char* buf, char* hub, int level)
680: {
681:     int devstart = 0,devend,num;
682:     char *spahub = "SPA_U_hub";
683:     char *camelspahub = "spaUHub";
684:     char *device;
685:     char name[128];
686:     int type = -1;
687:     unsigned int i = 0;
688:     unsigned int count = 0;
689:     size_t length = strlen(buf);
690:
691:     device = NULL;
692:     if(buf[0] == '\0')
693:         return type;
694:     while(i < length)
695:     {
696:         devstart = 0;
697:         devend = 0;
698:         for(i = count; i < length; i++)
699:         {
700:             if(strncmp(&buf[i],spahub,strlen(spahub))== 0 ||
701:                strncmp(&buf[i],camelspahub,strlen(camelspahub))== 0)
702:             {
703:                 devstart = i - 1;
704:                 if(level >= 4)
705:                     printf("devstart is %d \n",devstart);
706:                 type = 0;
707:             }
708:             if(devstart > 0 && strncmp(&buf[i],">",1) == 0)
709:             {
710:                 devend = i + 1;
711:                 if(level >= 4)
712:                     printf("devend is %d \n",devend);
713:                 device = (char*)malloc(1+devend-devstart);
714:                 if (device == NULL)
715:                     return -1;
716:                 memcpy(device,&buf[devstart],devend-devstart);
717:                 device[devend-devstart] = '\0';

```

```

717:         if(level >= 4)
718:             printf("SPA_U_HUB chunk is %s \n",device);
719:         break;
720:     }
721:     if(i+1 == strlen(buf))
722:         return -1;
723: }
724:     num = GetAttribute(device,spahub,name,sizeof(name),level);
725: if (device != NULL)
726:     free(device);
727:     if(num < 1)
728:     {
729:         return -1;
730:     }
731:     else
732:     {
733:         memcpy(hub,name,strlen(name)+1);
734:         break;
735:     }
736: }
737: return type;
738: }

```

File: sdm/dm/Makefile.uclinux

```
1: ifndef PETALINUX
2: $(error You must source the petalinux/settings.sh script before working with PetaLinux)
3: endif
4:
5: # Point to default PetaLinux root directory
6: ifndef ROOTDIR
7: ROOTDIR=$(PETALINUX)/software/petalinux-dist
8: endif
9:
10: PATH:=$(PATH):$(ROOTDIR)/tools
11:
12: UCLINUX_BUILD_USER = 1
13: -include $(ROOTDIR)/.config
14: -include $(ROOTDIR)/$(CONFIG_LINUXDIR)/.config
15: LIBCDIR = $(CONFIG_LIBCDIR)
16: -include $(ROOTDIR)/config.arch
17: ROMFSDIR=$(ROOTDIR)/romfs
18: ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh
19:
20: APP = dm
21: MONITOR_APP = dm_monitor
22:
23: # Add any other object files to this list below
24: APP_OBJS = DM.o Parse.o xTEDSLibrary.o xTEDSLibraryList.o xTEDSParameters.o
Subscription.o SubscriptionList.o backupDMList.o xTEDSSegmentBuilder.o DMUtils.o
ProviderSubscriptionList.o ProviderSubscription.o
25: MONITOR_APP_OBJS=dm_monitor.o
26:
27: FLTFLAGS+=-s 2097152
28: export FLTFLAGS
29:
30: LDLIBS += -lSDM -lpthread -lstdc++
31: LDFLAGS += -L../common/
32: all: $(APP) $(MONITOR_APP)
33:
34: $(APP): $(APP_OBJS)
35: $(CXX) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)
36: cp $@ ../FLIGHT_BINARIES/
37:
```

```

38: $(MONITOR_APP): $(MONITOR_APP_OBJS)
39: $(CXX) $(LDFLAGS) -o $$ $(MONITOR_APP_OBJS) $(LDLIBS)
40:
41: clean:
42: -rm -f $(APP) $(MONITOR_APP) *.elf *.gdb *.o
43:
44: romfs:
45: $(ROMFSINST) $(APP) /bin/$(APP)
46: $(ROMFSINST) $(MONITOR_APP) /bin/$(MONITOR_APP)
47:
48: %.o: %.cpp
49: $(CXX) -c $(CXXFLAGS) -o $$ $<
50:
51:
52: # Targets for the required .config files - if they don't exist, the tree isn't
53: # configured. Tell the user this, how to fix it, and exit.
54: ${ROOTDIR}/config.arch ${ROOTDIR}/.config:
55: @echo "Error: You must configure the PetaLinux tree before compiling your application"
56: @echo ""
57: @echo "Change directory to ../../petalinux-dist and 'make menuconfig' or 'make xconfig'"
58: @echo ""
59: @echo "Once the tree is configured, return to this directory, and re-run make."
60: @echo ""
61: @exit -1
62:

```

File: sdm/dm/xTEDSLibraryList.h

```
1: #ifndef __SDM_XTEDS_LIBRARY_LIST_H_
2: #define __SDM_XTEDS_LIBRARY_LIST_H_
3:
4: //The xTEDSLibraryList does not 'own' its items, it is just a way for items to reference other items in
the tree
5:
6: #include "xTEDSLibrary.h"
7: #include "../common/message/SDMComponent_ID.h"
8: #include <stdio.h>
9:
10: struct xTEDSLibraryListNode
11: {
12: xTEDSLibrary* data;
13: struct xTEDSLibraryListNode* next;
14: };
15:
16:
17:
18: The xTEDSLibraryList class is a linked list of xTEDSLibrary nodes, used by the DataManager to
represent
19: registered devices and applications in the SDM. xTEDSLibrary nodes are dynamically allocated
when a
20: free position in the list doesn't exists, but are not de-allocated when not used. Instead they are marked
21: as unavailable, which allows the position to be reused.
22:
23:
24:
25:
26:
27: class xTEDSLibraryList
28: {
29: public:
30: xTEDSLibraryList();
31: xTEDSLibraryList(const xTEDSLibraryList&);
32: ~xTEDSLibraryList();
33: bool GetDeviceName(const SDMComponent_ID& sdmCompId, char* pBuffer, unsigned int
uiBufferSize);
```

```
34: void addLibrary(xTEDSLibrary*);
35: int size();
36: xTEDSLibraryList& operator=(const xTEDSLibraryList&);
37: struct xTEDSLibraryListNode* head;      //Head of the list
38: struct xTEDSLibraryListNode* tail;     //Tail of the list
39: };
40:
41: #endif
```


File: sdm/dm/Makefile

```
1: # Makefile for client/server TCP
2: include ../Makefile.common
3: include ../$(MAKEFILE_DEFS)
4:
5: DMBUILDFLAGS=$(DMBACKUPFLAGS) $(DMMERGEFLAGS) $(DMFAKEBACKUPFLAGS)
6:
7: .PHONY:    clean distclean
8:
9: dm: dm.o    Parse.o xTEDSLibrary.o xTEDSLibraryList.o xTEDSParameters.o Subscription.o
SubscriptionList.o backupDMList.o xTEDSSegmentBuilder.o DMUtils.o ProviderSubscription.o
ProviderSubscriptionList.o
10: $(CXX) $(CXXFLAGS) -L../common -static -o $@ $^ -lSDM -lpthread
11:
12: dm_monitor: dm_monitor.cpp
13: $(CXX) $(CXXFLAGS) -L../common -static -o $@ $^ -lSDM -lpthread
14:
15: dm.o:  DM.cpp Parse.h
16: $(CXX) $(CXXFLAGS) $(DMBUILDFLAGS) -c -o $@ $<
17:
18: Parse.o: Parse.cpp Parse.h
19: $(CXX) $(CXXFLAGS) $(DMBUILDFLAGS) -c $<
20:
21: xTEDSLibrary.o:  xTEDSLibrary.cpp xTEDSLibrary.h
22: $(CXX) $(CXXFLAGS) -c $<
23:
24: xTEDSLibraryList.o:  xTEDSLibraryList.cpp xTEDSLibraryList.h
25: $(CXX) $(CXXFLAGS) -c $<
26:
27: xTEDSParameters.o:  xTEDSParameters.cpp xTEDSParameters.h
28: $(CXX) $(CXXFLAGS) -c $<
29:
30: Subscription.o: Subscription.cpp Subscription.h
31: $(CXX) $(CXXFLAGS) -c $<
32:
33: SubscriptionList.o: SubscriptionList.cpp SubscriptionList.h
34: $(CXX) $(CXXFLAGS) -c $<
35:
36: backupDMList.o:  backupDMList.cpp backupDMList.h
37: $(CXX) $(CXXFLAGS) -c $<
```

38:
39: xTEDSSegmentBuilder.o: xTEDSSegmentBuilder.cpp xTEDSSegmentBuilder.h
40: \$(CXX) \$(CXXFLAGS) -c \$<
41:
42: DMUtils.o: DMUtils.cpp DMUtils.h
43: \$(CXX) \$(CXXFLAGS) -c \$<
44:
45: ProviderSubscription.o: ProviderSubscription.cpp ProviderSubscription.h
46: \$(CXX) \$(CXXFLAGS) -c \$<
47:
48: ProviderSubscriptionList.o: ProviderSubscriptionList.cpp ProviderSubscriptionList.h
49: \$(CXX) \$(CXXFLAGS) -c \$<
50:
51: clean:
52: rm -f *.o *~ SDMMessages*
53:
54: distclean: clean
55: rm -f dm dm_monitor

File: sdm/dm/Subscription.cpp

```
1: #include "Subscription.h"
2: #include <string.h>
3: #include <stdlib.h>
4: #include <sys/socket.h>
5: #include <netinet/in.h>
6: #include <arpa/inet.h>
7: extern "C"
8: {
9: #include "../common/MemoryUtils.h"
10: }
11:
12:
Subscription::Subscription():m_SubMessageID(),m_bInUse(false),m_iID(0),m_strItemName(NULL),m_
strQualList(NULL),m_idSubscriber(),m_idSource(),m_strDevice(NULL),m_strInterface(NULL),m_bHas
Items(false)
13: {
14: }
15:
16:
Subscription::Subscription(const Subscription&
b):m_SubMessageID(),m_bInUse(false),m_iID(0),m_strItemName(NULL),m_strQualList(NULL),m_idS
ubscriber(),m_idSource(),m_strDevice(NULL),m_strInterface(NULL),m_bHasItems(false)
17: {
18: if(m_strItemName!=NULL) free(m_strItemName);
19: m_strItemName = SDM_strdup(b.m_strItemName);
20: if(m_strQualList!=NULL) free(m_strQualList);
21: m_strQualList = SDM_strdup(b.m_strQualList);
22: if(m_strDevice!=NULL) free(m_strDevice);
23: m_strDevice = SDM_strdup(b.m_strDevice);
24: if(m_strInterface!=NULL) free(m_strInterface);
25: m_strInterface = SDM_strdup(b.m_strInterface);
26: m_idSubscriber = b.m_idSubscriber;
27: m_bInUse = b.m_bInUse;
28: m_iID = b.m_iID;
29: m_bHasItems = b.m_bHasItems;
30: }
31:
32: Subscription::~Subscription()
33: {
34: if(m_strItemName!=NULL) free(m_strItemName);
35: if(m_strQualList!=NULL) free(m_strQualList);
```

```

36: if(m_strDevice!=NULL) free(m_strDevice);
37: if(m_strInterface!=NULL) free(m_strInterface);
38: }
39:
40: void Subscription::setAddress(char* new_IP)
41: {
42: m_idSubscriber.setAddress(inet_addr(new_IP));
43: }
44:
45: void Subscription::setPort(unsigned short new_port)
46: {
47: m_idSubscriber.setPort(new_port);
48: }
49:
50: void Subscription::setmID(int new_mID)
51: {
52: m_SubMessageID = new_mID;
53: }
54:
55: void Subscription::setInuse(bool new_inuse)
56: {
57: m_bInUse = new_inuse;
58: }
59:
60: void Subscription::setID(int new_ID)
61: {
62: m_iID = new_ID;
63: }
64:
65: void Subscription::setItemName(const char* new_itemname)
66: {
67: if(m_strItemName!=NULL) free(m_strItemName);
68: m_strItemName = SDM_strdup(new_itemname);
69: }
70:
71: void Subscription::setQuallist(const char* new_quallist)
72: {
73: if(m_strQualList!=NULL) free(m_strQualList);
74: m_strQualList = SDM_strdup(new_quallist);
75: }
76:

```

```

77: void Subscription::setItems(bool new_items)
78: {
79:     m_bHasItems = new_items;
80: }
81:
82: void Subscription::setDestination(const SDMComponent_ID& new_destination)
83: {
84:     m_idSubscriber = new_destination;
85: }
86:
87: void Subscription::setSource(const SDMComponent_ID& new_source)
88: {
89:     m_idSource = new_source;
90: }
91:
92: void Subscription::setDevice(const char* new_device)
93: {
94:     if(m_strDevice!=NULL) free(m_strDevice);
95:     m_strDevice = SDM_strdup(new_device);
96: }
97:
98: void Subscription::setInterface(const char* new_interface)
99: {
100:     if(m_strInterface!=NULL) free(m_strInterface);
101:     m_strInterface = SDM_strdup(new_interface);
102: }
103:
104: char* Subscription::getAddress(void) const
105: {
106:     char* addr;
107:     struct in_addr inaddr;
108:     inaddr.s_addr = m_idSubscriber.getAddress();
109:     addr = inet_ntoa(inaddr);
110:     return addr;
111: }
112:
113: unsigned short Subscription::getPort(void) const
114: {
115:     return m_idSubscriber.getPort();
116: }
117:

```

```

118: int Subscription::getmID(void) const
119: {
120:     return m_SubMessageID.getInterfaceMessagePair();
121: }
122:
123: /*
124: -- Now inline --
125:
126: int Subscription::getID(void) const
127: {
128:     return m_iID;
129: }
130:
131: SDMComponent_ID Subscription::getDestination(void) const
132: {
133:     return m_idSubscriber;
134: }
135:
136: SDMComponent_ID Subscription::getSource(void) const
137: {
138:     return m_idSource;
139: }
140:
141: const char* Subscription::getDevice(void) const
142: {
143:     return m_strDevice;
144: }
145:
146: const char* Subscription::getInterface(void) const
147: {
148:     return m_strInterface;
149: }
150:
151: const char* Subscription::getItemName(void) const
152: {
153:     return m_strItemName;
154: }
155:
156: const char* Subscription::getQuallist(void) const
157: {
158:     return m_strQualList;

```

```

159: }
160:
161: bool Subscription::getInuse(void) const
162: {
163:     return m_bInUse;
164: }
165:
166: bool Subscription::getItems(void) const
167: {
168:     return m_bHasItems;
169: }
170: */
171:
172: Subscription& Subscription::operator=(const Subscription& b)
173: {
174:     if(m_strItemName!=NULL) free(m_strItemName);
175:     m_strItemName = SDM_strdup(b.m_strItemName);
176:     if(m_strQualList!=NULL) free(m_strQualList);
177:     m_strQualList = SDM_strdup(b.m_strQualList);
178:     m_bInUse = b.m_bInUse;
179:     m_iID = b.m_iID;
180:     m_bHasItems = b.m_bHasItems;
181:     if(m_strDevice!=NULL) free(m_strDevice);
182:     m_strDevice = SDM_strdup(b.m_strDevice);
183:     if(m_strInterface!=NULL) free(m_strInterface);
184:     m_strInterface = SDM_strdup(b.m_strInterface);
185:     m_idSubscriber = b.m_idSubscriber;
186:     return *this;
187: }

```

File: sdm/dm/ProviderSubscriptionList.h

```
1: #ifndef _SDM_PROVIDER_SUBSCRIPTION_LIST_H_
2: #define _SDM_PROVIDER_SUBSCRIPTION_LIST_H_
3:
4: #include "../common/message/SDMComponent_ID.h"
5: #include "../common/message/SDMMessage_ID.h"
6:
7: #include "ProviderSubscription.h"
8:
9: // #define DEBUG_SUB_LIST    1
10:
11: /*
12:  * Provider subscription list allows the DM to keep track of all applications subscribed
13:  * to all device notifications. If an application is reported to have failed from the PM
14:  * the DM can cancel all of the subscriptions to the device notifications. This is required
15:  * to free up resources on ASIM who may not receive a cancellation.
16:  *
17:  */
18:
19: class ProviderSubscriptionListNode
20: {
21: public:
22: ProviderSubscriptionListNode() : Data(), pNext(NULL) {}
23: ProviderSubscriptionListNode(const ProviderSubscriptionListNode&);
24: ProviderSubscriptionListNode& operator=(const ProviderSubscriptionListNode&);
25: ProviderSubscription Data;
26: ProviderSubscriptionListNode* pNext;
27: };
28:
29: class ProviderSubscriptionList
30: {
31: public:
32: ProviderSubscriptionList();
33: ~ProviderSubscriptionList();
34: ProviderSubscriptionList(const ProviderSubscriptionList&);
35: ProviderSubscriptionList& operator=(const ProviderSubscriptionList&);
36:
37: void Add(const SDMComponent_ID& ProviderId, const SDMComponent_ID& SubscriberId, const
SDMMessage_ID& MessageId);
38: void Delete(const SDMComponent_ID& SubscriberId, const SDMMessage_ID& MessageId);
```



```
39: void SubscriberFinish(const SDMComponent_ID& SubscriberId);
40: void ProviderFinish(const SDMComponent_ID& ProviderId);
41: void PrintList(const char* strFunction);
42: void DeleteAll(const SDMComponent_ID& SubscriberId, bool bUseProviderId);
43: private:
44: void Add(ProviderSubscriptionListNode* pNode);
45: void DeleteList();
46: ProviderSubscriptionListNode* m_pHead;
47: };
48:
49: #endif
```

File: sdm/dm/dm_monitor.cpp

```
1: #include <stdio.h>
2: #include <unistd.h>
3: #include <sys/types.h>
4: #include <signal.h>
5: #include <stdlib.h>
6: #include <arpa/inet.h>
7: #include <netinet/in.h>
8: #include <sys/socket.h>
9: #include <sys/wait.h>
10: #include <string.h>
11: #include "../common/message/SDMReady.h"
12: #include "../common/message/SDMReqReg.h"
13: #include "../common/message/SDMRegInfo.h"
14: #include "../common/message/SDMCommand.h"
15: #include "../common/MessageManager/MessageManager.h"
16: #include "../common/TCPcom.h"
17:
18: #define DEBUG_DM_MONITOR 1
19:
20: #define NUM_HEARTBEAT_TRIES 12 /*Number of times to miss a response before restarting
the DM*/
21:
22: // TODO:
23: const unsigned int RESET_SECURITY_KEY = 1234;
24: const char* STR_RESET_DEVICE_NAME = "PowerController";
25: const char* STR_RESET_DEVICE_INTERFACE = "PowerStatusInterface";
26: const char* STR_RESET_DEVICE_ITEM = "SystemWarmBoot";
27:
28: SDMComponent_ID idResetDevice;
29: SDMMessage_ID idCmdReset;
30:
31: bool bResetDeviceFound = false;
32: const int ID_RESET_COMMAND = 3;
33:
34: int StartDM(char **);
35: void SigIntHandler(int);
36: void DMFailure();
37: void SendReqReg();
38: void RegInfoHandler(const char* msgBuf);
```

```

39:
40: int dm_pid = -1;    //Process id number of the Data Manager
41:
42: int main(int argc, char ** argv)
43: {
44:     int result = -1;
45:     int miss_count = 0;
46:     char buf[BUFSIZE];
47:     SDMReady heartbeat;
48:     MessageManager mm;
49:
50:     if (argc == 1)
51:     {
52:         printf("Usage: \n%s dm_process [dm_process_args...] \n", argv[0]);
53:         return -1;
54:     }
55:
56:     // Set up the command line for the child process
57:     char** nargv = new char*[argc];
58:     for (int i = 1; i < argc; ++i)
59:         nargv[i-1] = argv[i];
60:     nargv[argc-1] = NULL;
61:
62:     //Set monitor process address
63:     heartbeat.destination.setSensorID(0);
64:     heartbeat.destination.setAddress(inet_addr("127.0.0.1"));
65:     heartbeat.destination.setPort(PORT_DM_MONITOR);
66:
67:     //Set DM address
68:     DataManager.setAddress(inet_addr("127.0.0.1"));
69:     DataManager.setPort(PORT_DM);
70:     DataManager.setSensorID(0);
71:
72:     signal(SIGINT, SigIntHandler);
73:     dm_pid = StartDM(nargv);
74:     mm.Async_Init(PORT_DM_MONITOR);
75:
76:     //If fork/exec was successful
77:     if (dm_pid > 0)
78:     {
79:         unsigned char ucType;

```

```

80:     bool bHeartbeatReceived = false;
81:     //Allow the DM to get started up
82:     sleep(HEARTBEAT_INTERVAL);
83:
84:     // Request the reset device
85:     SendReqReg();
86:     while (1)
87:     {
88:         bHeartbeatReceived = false;
89:         //Send heartbeats via UDP
90:         heartbeat.SendTo(DataManager);
91:         sleep(HEARTBEAT_INTERVAL);
92:
93:         //If Data Manager quit
94:         if (waitpid(-1,&result,WNOHANG) == dm_pid)
95:         {
96:             printf(" Monitor: DM failed... \n");
97:             DMFailure();
98:         }
99:         while (mm.IsReady())
100:         {
101:             ucType = mm.GetMessage(buf);
102:             switch(ucType)
103:             {
104:                 case SDM_Ready:
105:                 {
106:                     miss_count = 0;
107:                     bHeartbeatReceived = true;
108:                 }
109:                 break;
110:                 case SDM_RegInfo:
111:                 {
112:                     RegInfoHandler(buf);
113:                 }
114:                 break;
115:                 default:
116:                     printf(" Monitor: Unexpected message 0x%hhx \n", ucType);
117:                     break;
118:             }
119:         }
120:

```

```

121:         if (!bHeartbeatReceived)
122:         {
123:             //If no messages were received
124:             miss_count++;
125:             if (miss_count >= NUM_HEARTBEAT_TRIES)
126:             {
127:                 printf(" Monitor: DM unresponsive... \n");
128:                 DMFailure();
129:             }
130:         }
131:     }
132: }
133: else
134: {
135:     printf(" Monitor: Error starting the DM (%d). \n", dm_pid);
136:     return -1;
137: }
138:
139: return 0;
140: }
141:
142: void SendReqReg()
143: {
144:     SDMReqReg msgReqReg;
145:
146:     msgReqReg.destination.setPort(PORT_DM_MONITOR);
147:     strncpy(msgReqReg.device, STR_RESET_DEVICE_NAME, sizeof(msgReqReg.device));
148:     strncpy(msgReqReg.interface, STR_RESET_DEVICE_INTERFACE,
149:             sizeof(msgReqReg.interface));
150:     strncpy(msgReqReg.item_name, STR_RESET_DEVICE_ITEM,
151:             sizeof(msgReqReg.item_name));
152:     msgReqReg.id = ID_RESET_COMMAND;
153:     msgReqReg.reply = SDM_REQREG_CURRENT_AND_FUTURE;
154:     msgReqReg.Send();
155: #ifdef DEBUG_DM_MONITOR
156:     printf(" Monitor: ReqReg sent for reset device. \n");
157: #endif
158: }
159: void RegInfoHandler(const char* msgBuf)

```

```

160: {
161:     SDMRegInfo msgRegInfo;
162:     long lResult = 0;
163:     if ((lResult = msgRegInfo.Unmarshal(msgBuf)) == SDM_NO_FURTHER_DATA_PROVIDER)
164:         return;
165:     else if (lResult < 0)
166:     {
167:         printf(" Monitor: received invalid SDMRegInfo message. \n");
168:         return;
169:     }
170:
171:     if (msgRegInfo.id == ID_RESET_COMMAND && msgRegInfo.type ==
SDM_REGINFO_REGISTRATION)
172:     {
173:         idResetDevice = msgRegInfo.source;
174:         idCmdReset = msgRegInfo.msg_id;
175:         bResetDeviceFound = true;
176: #ifdef DEBUG_DM_MONITOR
177:         printf(" Monitor: Reset device found. \n");
178: #endif
179:     }
180:     else if (msgRegInfo.id == ID_RESET_COMMAND && msgRegInfo.type ==
SDM_REGINFO_CANCELLATION)
181:     {
182:         bResetDeviceFound = false;
183: #ifdef DEBUG_DM_MONITOR
184:         printf(" Monitor: Reset device cancelled. \n");
185: #endif
186:     }
187: }
188:
189: void DMFailure()
190: {
191: #ifdef DEBUG_DM_MONITOR
192:     printf(" Monitor: %s() \n", __FUNCTION__);
193: #endif
194:     if (!bResetDeviceFound)
195:     {
196:         printf(" Monitor: Reset device not found. \n");
197:         return;
198:     }

```

```

199:
200:   SDMCommand msgCommand;
201:
202:   msgCommand.command_id = idCmdReset;
203:   msgCommand.source = idResetDevice;
204:   PUT_UINT(msgCommand.data, RESET_SECURITY_KEY);
205:   msgCommand.length = sizeof(RESET_SECURITY_KEY);
206:
207:   msgCommand.Send(idResetDevice);
208:
209:   sleep(1);
210:   fflush(NULL);
211:   exit(EXIT_FAILURE);
212: }
213:
214: int StartDM(char ** argv)
215: {
216:     int pid;
217:
218:     pid = vfork();
219:     //Child Process
220:     if (pid == 0)
221:     {
222:         //Start the Data Manager
223:         if (execvp(argv[0],argv) < 0)
224:         {
225:             printf(" Monitor: Error exec'ing the DM. \n");
226:             exit(-1);
227:         }
228:         //Here to make the compiler happy, never runs however
229:         return 0;
230:     }
231:     //Parent Process
232:     else
233:         return pid;
234: }
235:
236: void SigIntHandler(int sig_num)
237: {
238:     int result = 0;
239:     if (kill (SIGINT, dm_pid) == 0)

```

```
240:     wait (&result);
241:  else
242:     waitpid(-1,&result,WNOHANG);
243:
244:  exit(EXIT_SUCCESS);
245: }
246:
```


File: sdm/dm/backupDMList.h

```
1:
2:
3: The backupDMList class is used to keep track of all registered backup DataManagers. Backup
DataManagers receive
4: all copies of SDMxTEDS and SDMCcancelxTEDS messages. This class helps facilitate sending those
messages out.
5: This list is built when SDMReady messages arrive from the backup DMs as heartbeat messages.
6:
7:
8: #ifndef __SDM_BACKUP_DM_LIST_H_
9: #define __SDM_BACKUP_DM_LIST_H_
10:
11: #include <stdio.h>
12: #include "../common/message/SDMComponent_ID.h"
13: #include "../common/message/SDMmessage.h"
14:
15: #define DEBUG_DM_BACKUP_LIST 1
16:
17: class DMBackupListNode
18: {
19: public:
20: DMBackupListNode() : m_idAddress(), m_pNext(NULL) {}
21: DMBackupListNode(const DMBackupListNode&);
22: DMBackupListNode& operator=(const DMBackupListNode&);
23: SDMComponent_ID m_idAddress;
24: class DMBackupListNode* m_pNext;
25: };
26:
27: class DMBackupList
28: {
29: public:
30: DMBackupList();
31: DMBackupList(const DMBackupList&);
32: DMBackupList& operator=(const DMBackupList&);
33: ~DMBackupList();
34:
```

```
35: bool AddNode(const SDMComponent_ID& NewDmId);
36: bool AddIfNew(const SDMComponent_ID& DmId);
37: void SendMessageToAll(SDMmessage& Message);
38: bool SendMessageTo(const SDMComponent_ID& DmId, SDMmessage& Message);
39: void PrintList();
40:
41: private:
42: DMBackupListNode* m_pHead;
43: DMBackupListNode* m_pLastAccess;
44:
45: void DeleteList();
46: bool ListContains(const SDMComponent_ID& Id);
47: DMBackupListNode* Find(const SDMComponent_ID& Id);
48: };
49:
50: #endif
```

File: sdm/dm/DMUtils.cpp

```
1: // Extra utility functions used by the Data Manager
2:
3: #include <string.h>
4: #include <stdio.h>
5: #include <unistd.h>
6: #include <stdlib.h>
7: #include <sys/types.h>
8: #include <sys/socket.h>
9: #include <sys/ioctl.h>
10: #include <sys/stat.h>
11: #include <fcntl.h>
12: #include <netinet/in.h>
13: #include <arpa/inet.h>
14:
15: #ifdef __VXWORKS__
16: #include <sockLib.h>
17: #include <ioLib.h>
18: #endif
19:
20: #include "../common/message/SDMxTEDS.h"
21: #include "DMUtils.h"
22:
23: #ifdef SEND_IMA
24: #include <netinet/in.h>
25: #include <netspwnp/spwnp.h>
26: #endif
27:
28:
29: /*
30: Description:
31:     Is the xTEDS from an ASIM (ie negative (MSB set))?
32:
33: Input:
34:     processID - ASIM PID (ie negative, less than zero)
35:
36: Output:
37:     true - if processID is negative
38:     false - if processID is non-negative
39:
```

```

40: Changed:
41:     None
42:
43: Written by: Kelly Wheeler
44: Date:      04/17/07
45: */
46: bool IsPIDFromASIM (long processID)
47: {
48:     if (processID < 0)
49:         return true;
50:     else
51:         return false;
52: }
53:
54:
55: /*
56: Description:
57:     Convert a processID to a filename.
58:
59: Input:
60:     processID - ASIM PID (ie negative, less than zero)
61:     fileName - pointer for output conversion
62:
63: Output:
64:     true - if fileName is valid
65:     false - if fileName is invalid
66:
67: Changed:
68:     fileName as a string, or null if failed.
69:
70: Written by: Kelly Wheeler
71: Date:      04/17/07
72: */
73: bool PIDToFileName (long processID, char fileName[])
74: {
75:     if (IsPIDFromASIM (processID))
76:     {
77:         sprintf (fileName, "./xTEDS/%ld.xml", processID * -1); // convert processID to filename
78:         return true;
79:     }
80:     else

```

```

81:     return false;
82: }
83:
84:
85: /*
86: Description:
87:     Write an xTEDS to the file system using processID as the name
88:
89: Input:
90:     long processID - the ASIM's PID type to be used as name of the stored xTEDS file
91:     char* xTEDSIn - valid xTEDS to store in filesystem
92:
93: Output:
94:     error:
95:     0 - Success
96:     -1 - PID is not from ASIM
97:     -2 - Unable to open xTEDS file of given PID
98:     -3 - No bytes written to file
99:
100:    Changed:
101:        - new file created in xTEDS directory using the input's processID as the file name. ie. -34 =
102:        34.xml
103:    Written by: Kelly Wheeler
104:    Date:      04/17/07
105: */
106: int Store_xTEDS (long processID, char* xTEDSIn)
107: {
108:     char fileName[FILENAME_SIZE];                // xTEDS filename with .xml
extension
109:     int bytesWritten;                            // number of bytes written
110:     int fd = 0;                                  // file descriptor
111:     char endOfLine = '\n';
112:
113:
114:     if (!PIDToFileName (processID, fileName))    // convert processID to filename
115:     {
116:         printf (" \tStore_xTEDS: processID is not from ASIM \n \n");
117:         return -1;                                // return error -1
118:     }
119: #ifdef WIN32

```

```

120:   fd = open (fileName, O_RDWR | O_CREAT);           // open storage file for writing
121: #else
122:   fd = open (fileName, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
123: #endif
124:
125:   if(fd == -1)                                     // Can't open filename?
126:   {
127:       perror("Store_xTEDS: Unable to open ASIM xTEDS file! ");
128:       return -2;                                   // return error -2
129:   }
130:
131:   bytesWritten = write(fd, xTEDSIn, strlen(xTEDSIn)); // write xTEDS to file
132:
133:   bytesWritten = write(fd, &endOfLine, 1);           // end the file
134:
135:   close(fd);                                         // close the file
136:
137:   if (bytesWritten < 0)                             // did anything actually get written?
138:       return -3;                                   // No - return -3
139:   else
140:       return 0;                                     // Yes - return true
141: }
142:
143:
144: /*
145:   Description:
146:       Read an xTEDS from the file system using processID as the name
147:
148:   Input:
149:       long processID - the ASIM's PID (ie negative, less than zero) type to be used as name of the
       stored xTEDS file
150:
151:   Output:
152:       0 - Success
153:       -1 - PID is not from ASIM
154:       -2 - Unable to open xTEDS file of given PID
155:
156:   Changed:
157:       char* xTEDSOut - xTEDS from filesystem, but not validated. Or NULL if unread.
158:
159:   Written by: Kelly Wheeler

```

```

160:   Date:      04/17/07
161: */
162: int Retrieve_xTEDS (long processID, char* xTEDSOut)
163: {
164:   char fileName[FILENAME_SIZE];           // xTEDS filename with .xml
extension
165:   int bytesRead;                          // number of bytes read
166:   int fd = 0;                             // file descriptor
167:
168:
169:   if (!PIDToFileName (processID, fileName)) // convert PID to filename
170:   {
171:     printf ("\tRetrieve_xTEDS: processID is not from ASIM \n \n");
172:     return -1;                             // return error -1
173:   }
174:
175:   fd = open(fileName,O_RDONLY);            // open the xTEDS file for reading
176:   if(fd == -1)                             // if open failed return error
177:   {
178:     perror("Unable to open ASIM xTEDS file! ");
179:     return -2;                             // return error -2
180:   }
181:
182:   bytesRead = read(fd, xTEDSOut, MAX_XTEDS_SIZE); // read the whole file at once
183:
184:   close(fd);                              // close the xTEDS file
185:   xTEDSOut[bytesRead] = '\0';              // end the buffer string
186:
187:   return 0;                               // return true
188: }
189:
190: #ifdef SEND_IMA
191: int SendIMA(unsigned char which, int debug)
192: {
193:   int sock;
194:   struct sockaddr_spwpnp sspwpnp;
195:
196:   struct hostent *he;
197:   struct in_addr *nm_addr;
198:   struct in_addr *lh_addr;
199:

```

```

200: struct spwnpnp_ima ima;
201:
202: unsigned long nm_addr_host;
203: unsigned long lh_addr_host;
204:
205: int sent;
206:
207: while ((he=gethostbyname("localhost.spacewire")) == NULL)
208: {
209:     if (debug >= 1)
210:     {
211:         printf("gethostbyname failed for localhost.spacewire. Retrying in 5 secs. \n");
212:     }
213:     sleep(5);
214: }
215: lh_addr = (struct in_addr*)he->h_addr;
216: lh_addr_host = ntohl(lh_addr->s_addr);
217:
218: if (debug >= 2)
219: {
220:     printf("Got localhost address %s \n", inet_ntoa(*lh_addr));
221: }
222:
223: while ((he=gethostbyname("networkmanager.spacewire")) == NULL)
224: {
225:     if (debug >= 1)
226:     {
227:         printf("gethostbyname failed for networkmanager.spacewire. Retrying in 5 secs. \n");
228:     }
229:     sleep(5);
230: }
231: nm_addr = (struct in_addr*)he->h_addr;
232: nm_addr_host = ntohl(nm_addr->s_addr);
233:
234: if (debug >= 2)
235: {
236:     printf("Got NetworkManager address %s \n", inet_ntoa(*nm_addr));
237: }
238:
239: /*
240:  * As defined, we translate an ip to a router id and router port:

```



```

241:  * 10.x.x.y --> xx = router id, y = router port
242:  *
243:  */
244:  memset(&sspwnp, 0, sizeof(sspwnp));
245:  sspwnp.sspwnp_family = AF_SPWPNP;
246:  sspwnp.sspwnp_router_id = htons((nm_addr_host >> 8) & 0xFFFF);
247:  sspwnp.sspwnp_router_port = (nm_addr_host & 0xFF);
248:
249:  sock = socket(PF_SPWPNP, SOCK_DGRAM, SPWPNPPROTO_ANY);
250:  if (sock < 0 )
251:  {
252:      perror("socket");
253:      return -1;
254:  }
255:
256:  ima.hdr.logical_addr = LogicalAddrAny;
257:  ima.hdr.protocol_id = ProtocolLl;
258:  ima.hdr.packet_type = PacketNetworkManager;
259:  ima.hdr.data_type = DataIma;
260:  ima.hdr.transaction_id = which;
261:  ima.hdr.data_select = ImaPrimary;
262:  ima.hdr.data_len = htons(sizeof(ima) - sizeof(ima.hdr) - sizeof(ima.csum));
263:  ima.ip = htonl(lh_addr_host);
264:  ima.port = htons(PORT_DM);
265:  ima.csum = 0;
266:
267:  printf("sending ima host 0x%x port %d size %d \n", ntohl(ima.ip), ntohs(ima.port), sizeof(ima));
268:
269:  sent = sendto(sock, &ima, sizeof(ima), 0, (struct sockaddr*)&sspwnp, sizeof(sspwnp));
270:  if (sent < 0)
271:  {
272:      perror("sendto");
273:      return -1;
274:  }
275:
276:  if (debug >= 2)
277:  {
278:      printf("Sent IMA \n");
279:  }
280:
281:  return 0;

```

```

282: }
283: #endif
284:
285: /* GetNodeAddress sets the IP address of the DataManager in the global Address_DM.
286:    Returns: unsigned long - The address of the current node, or zero if an error occurred
287: */
288: #ifdef WIN32
289: unsigned long GetNodeAddress(bool spacewire)
290: {
291:     hostent * localHost;
292:     char hostname[64];
293:     char * IPAddr;
294:     unsigned long Address = 0;
295:
296:     if (gethostname(hostname,sizeof(hostname)) < 0)
297:         return 0;
298:
299:     if ((localHost = gethostbyname(hostname)) == NULL)
300:         return 0;
301:
302:     IPAddr = inet_ntoa(*(struct in_addr *)localHost->h_addr_list[0]);
303:     Address = inet_addr(IPAddr);
304:     return Address;
305: }
306: #else
307:
308: #ifdef __uCLinux__
309: #warning "uCLinux GetNodeAddress"
310: unsigned long GetNodeAddress(bool spacewire)
311: {
312:     unsigned long Address = 0;
313:
314:     if (spacewire) {
315:         struct hostent *he;
316:
317:         while ((he=gethostbyname("localhost.spacewire")) == NULL) {
318:             sleep(1);
319:         }
320:
321:         memcpy(&Address, he->h_addr, sizeof(Address));
322:     }

```

```

323: else {
324:     struct ifreq ifr;
325:     struct sockaddr_in *sin = (struct sockaddr_in *)&ifr.ifr_addr;
326:     int sockfd;
327:
328:     bzero(&ifr, sizeof(ifr));
329:     if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
330:     {
331:         perror("socket()");
332:         return(0);
333:     }
334:     strcpy(ifr.ifr_name, "eth0");
335:     sin->sin_family = AF_INET;
336:     if(ioctl(sockfd, SIOCGIFADDR, &ifr) == 0)
337:     {
338:         Address = inet_addr(inet_ntoa(sin->sin_addr));
339:     }
340:     close(sockfd);
341: }
342: return Address;
343: }
344:
345: #else /* Linux, default */
346: unsigned long GetNodeAddress(bool spacewire)
347: {
348:     struct ifreq ifr;
349:     struct sockaddr_in *sin = (struct sockaddr_in *)&ifr.ifr_addr;
350:     int sockfd;
351:     unsigned long Address = 0;
352:
353:     memset(&ifr, 0, sizeof(ifr));
354:     if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
355:     {
356:         perror("socket()");
357:         return(0);
358:     }
359:
360: #ifdef __VXWORKS__
361:     strcpy(ifr.ifr_name, "rxg0");
362: #else
363:     strcpy(ifr.ifr_name, "eth0");

```

```

364: #endif
365:     sin->sin_family = AF_INET;
366:     if(ioctl(sockfd, SIOCGIFADDR, &ifr) == 0)
367:     {
368:         Address = inet_addr(inet_ntoa(sin->sin_addr));
369:     }
370:     close(sockfd);
371:     return Address;
372: }
373: #endif
374: #endif
375:
376: void CreatexTEDSDirectory()
377: {
378: #ifdef WIN32
379:     DWORD exists = GetFileAttributes(". \\xTEDS");
380:     if(exists == 0xFFFFFFFF)
381:         system("mkdir . \\xTEDS");           // make the xTEDS backup directory or fail if
exists
382: #else
383:     struct stat folder;
384:     if(stat("xTEDS",&folder) < 0)
385:         system("mkdir xTEDS");           // make the xTEDS backup directory or fail if
exists
386: #endif
387: }
388:

```

File: sdm/dm/ProviderSubscriptionList.cpp

```
1: #include <stdio.h>
2: #include "../common/message/SDMDeletesub.h"
3: #include "../common/message/SDMCancel.h"
4: #include "ProviderSubscriptionList.h"
5:
6: ProviderSubscriptionList::ProviderSubscriptionList() : m_pHead(NULL)
7: {
8: }
9:
10: ProviderSubscriptionList::~~ProviderSubscriptionList()
11: {
12: DeleteList();
13: }
14:
15: //
16: // Delete the entire list
17: void ProviderSubscriptionList::DeleteList()
18: {
19: ProviderSubscriptionListNode* pTemp;
20: for (ProviderSubscriptionListNode* pCur = m_pHead; pCur != NULL; )
21: {
22:     pTemp = pCur->pNext;
23:     delete pCur;
24:     pCur = pTemp;
25: }
26: m_pHead = NULL;
27: #ifdef DEBUG_SUB_LIST
28: PrintList(__FUNCTION__);
29: #endif
30: }
31:
32: //
33: // Add the indicated subscription
34: void ProviderSubscriptionList::Add(const SDMComponent_ID& ProviderId, const
SDMComponent_ID& SubscriberId, const SDMMMessage_ID& MessageId)
35: {
36: ProviderSubscriptionListNode* NewNode = new ProviderSubscriptionListNode();
37:
38: NewNode->Data.Set(ProviderId, SubscriberId, MessageId);
```

```

39:
40: Add(NewNode);
41: }
42:
43: //
44: // Add to the end of the list
45: void ProviderSubscriptionList::Add(ProviderSubscriptionListNode* pNode)
46: {
47: if (m_pHead == NULL)
48:     m_pHead = pNode;
49: else
50: {
51:     ProviderSubscriptionListNode* pCur = m_pHead;
52:     for (; pCur->pNext != NULL; pCur = pCur->pNext)
53:         ;
54:     pCur->pNext = pNode;
55: }
56: #ifdef DEBUG_SUB_LIST
57: PrintList(__FUNCTION__);
58: #endif
59: }
60:
61: //
62: // For the finished subscriber, cancel all subscriptions and delete the subscriber
63: // from the list.
64: void ProviderSubscriptionList::SubscriberFinish(const SDMComponent_ID& SubscriberId)
65: {
66: SDMDelatesub msgDelete;
67: SDMComponent_ID temp;
68:
69: for (ProviderSubscriptionListNode* pCur = m_pHead; pCur != NULL; pCur = pCur->pNext)
70: {
71:     temp = pCur->Data.GetSubscriber();
72:     //printf("Sub Add: 0x%lx Port: %li SID: %li \nComp Add: 0x%lx Port: %li SID: %li \n",
73:         //SubscriberId.getAddress(), SubscriberId.getPort(), SubscriberId.getSensorID(),
74:         //temp.getAddress(), temp.getPort(), temp.getSensorID());
75:     if (SubscriberId.getAddress() == temp.getAddress() && SubscriberId.getPort() ==
temp.getPort())
76:     {
77:         msgDelete.source = pCur->Data.GetProvider();
78:         msgDelete.destination = pCur->Data.GetSubscriber();

```

```

79:         msgDelete.msg_id = pCur->Data.GetMessage();
80: #ifdef DEBUG_SUB_LIST
81:     {
82:         char strProvider[128];
83:         char strSubscriber[128];
84:         char strMessage[128];
85:         msgDelete.source.IDToString(strProvider, sizeof(strProvider));
86:         msgDelete.destination.IDToString(strSubscriber, sizeof(strSubscriber));
87:         msgDelete.msg_id.IDToString(strMessage, sizeof(strMessage));
88:
89:         printf("Provider SubList SDMDelatesub sent: \n provider %s subscriber %s message %s
\n",
90:             strProvider, strSubscriber, strMessage);
91:     }
92: #endif
93:     msgDelete.Send();
94:     //printf("Sending Deletsub Msg \n");
95: }
96: }
97:
98: DeleteAll(SubscriberId, false /*Delete subscriber id*/);
99: }
100:
101: void ProviderSubscriptionList::ProviderFinish(const SDMComponent_ID& ProviderId)
102: {
103:     DeleteAll(ProviderId, true /*Delete provider id*/);
104: }
105:
106: //
107: // Delete the entry with the specified subscriber and message
108: void ProviderSubscriptionList::Delete(const SDMComponent_ID& SubscriberId, const
SDMMessage_ID& MessageId)
109: {
110:     ProviderSubscriptionListNode* pCur = m_pHead, *pPrev = NULL, *pTemp = NULL;
111:
112:     while (pCur != NULL)
113:     {
114:         if (pCur->Data.GetSubscriber() == SubscriberId && pCur->Data.GetMessage() ==
MessageId)
115:         {
116:             if (pPrev == NULL)
117:             {

```

```

118:         // We are deleting the head
119:         pTemp = pCur->pNext;
120:         delete m_pHead;
121:         pCur = m_pHead = pTemp;
122:     }
123:     else
124:     {
125:         // Otherwise, we are at a middle element
126:         pTemp = pCur->pNext;
127:         delete pCur;
128:         pCur = pPrev->pNext = pTemp;
129:     }
130: }
131: else
132: {
133:     pPrev = pCur;
134:     pCur = pCur->pNext;
135: }
136: }
137: #ifdef DEBUG_SUB_LIST
138:     PrintList(__FUNCTION__);
139: #endif
140: }
141:
142: //
143: // Delete all entries with the given SubscriberId
144: void ProviderSubscriptionList::DeleteAll(const SDMComponent_ID& compId, bool
bUseProviderId)
145: {
146:     ProviderSubscriptionListNode* pCur = m_pHead, *pPrev = NULL, *pTemp = NULL;
147:     SDMComponent_ID subscriber, temp2;
148:     SDMCancel cancelMsg;
149:     while (pCur != NULL)
150:     {
151:         subscriber = pCur->Data.GetSubscriber();
152:         temp2 = pCur->Data.GetProvider();
153:
154:         // printf("SubAdd: 0x%lx Port: %li ID: %li \nProvAdd: 0x%lx Port: %li SID: %li \nComp to
Add: 0x%lx Port: %li ID: %li \n",
155:         // subscriber.getAddress(), subscriber.getPort(), subscriber.getSensorID(),
156:         // temp2.getAddress(), temp2.getPort(), temp2.getSensorID(),

```



```

157: //      compId.getAddress(), compId.getPort(), compId.getSensorID());
158:      if ((bUseProviderId && pCur->Data.GetProvider() == compId)
159:          || (!bUseProviderId && subscriber.getAddress() == compId.getAddress() &&
subscriber.getPort() == compId.getPort()))
160:      {
161:          // Perform a delete
162:          //printf("Deleteing Subscription.... \n");
163:          if (pPrev == NULL)
164:          {
165:              // We are deleting the head
166:              pTemp = pCur->pNext;
167:              delete m_pHead;
168:              pCur = m_pHead = pTemp;
169:          }
170:          else
171:          {
172:              // Otherwise, we are at a middle element
173:              pTemp = pCur->pNext;
174:              delete pCur;
175:              pCur = pPrev->pNext = pTemp;
176:          }
177:      }
178:      else
179:      {
180:          pPrev = pCur;
181:          pCur = pCur->pNext;
182:      }
183:  }
184: #ifdef DEBUG_SUB_LIST
185:  PrintList(__FUNCTION__);
186: #endif
187: }
188:
189: void ProviderSubscriptionList::PrintList(const char* strFunction)
190: {
191:     printf("*****Provider Subscription List***** \n");
192:     printf("Called function %s() \n", strFunction);
193:     char strProvider[128];
194:     char strSubscriber[128];
195:     char strMessage[128];
196:     for (ProviderSubscriptionListNode* pCur = m_pHead; pCur != NULL; pCur = pCur->pNext)

```

```
197:  {
198:      pCur->Data.GetSubscriber().IDToString(strSubscriber, sizeof(strSubscriber));
199:      pCur->Data.GetProvider().IDToString(strProvider, sizeof(strProvider));
200:      pCur->Data.GetMessage().IDToString(strMessage, sizeof(strMessage));
201:      printf("  entry:  provider %s subscriber %s message %s \n", strProvider, strSubscriber,
strMessage);
202:  }
203:  printf("***** \n");
204: }
205:
206:
207:
```

File: sdm/dm/SubscriptionList.h

```
1:
/*****
*****
2:
3: The SubscriptionList class is a linked list of SubscriptionListNode items which are used to add and
remove various
4: subscriptions specific to the DataManager module of the SDM. Namely, this list holds subscription
information for
5: DataManager notification subscriptions, SDMReqReg query subscriptions (when
SDM_REQREG_CURRENT_AND_FUTURE and
6: SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS search types are used), and
SDMSearch query subscriptions (when the
7: SDM_SEARCH_CURRENT_AND_FUTURE search type is used.
8:
9:
*****/
10: #ifndef __SDM_SUBSCRIPTION_LIST_H_
11: #define __SDM_SUBSCRIPTION_LIST_H_
12:
13: //The SubscriptionList does not 'own' its items, it is just a way for items to reference other items in the
tree
14:
15: #include "Subscription.h"
16: #include "../common/message/SDMComponent_ID.h"
17: #include <stdio.h>
18:
19:
20: #define REQ_REG_FUTURE 0x401
21: #define REQ_REG_CANCELLATION 0x402
22: #define VAR_REQ_REPLY 0x403
23: #define SEARCH_REPLY 0x404
24:
25: #define SUB_CURRENT_AND_FUTURE 5 // Corresponds to SDM_REQREG_* macros
with same value for list request
26: #define SUB_CURRENT_FUTURE_AND_CANCELLATIONS 7 // Corresponds to
SDM_REQREG_* macros with same value for list request
27: #define SUB_CANCEL 11 // Corresponds to SDM_REQREG_* macros with same
value for list request
28:
29: /* Linked list node */
```

```

30: struct SubscriptionListNode
31: {
32: Subscription* data;
33: struct SubscriptionListNode* next;
34: };
35:
36: class SubscriptionList
37: {
38: public:
39: SubscriptionList();
40: SubscriptionList(const SubscriptionList&);
41: ~SubscriptionList();
42: void addSubscription(Subscription*);
43: SubscriptionListNode* addSubscription(const SDMComponent_ID& SubscriberId, int mID, int
debug);
44: bool addOrRemoveSubscription(int reply, const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* device, const char* interface, const char* itemname, const
char* quallist, int ID, int mID, int debug);
45: bool addReqRegSubscription(int reply, const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* device, const char* interface, const char* itemname, const
char* quallist, int ID, int& error, int debug);
46: bool removeReqRegSubscription(int reply, const SDMComponent_ID& SubscriberId, int debug);
47: bool removeReqRegSubscription(int reply, const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* device, const char* interface, const char* itemname, const
char* quallist, int debug);
48: bool addSearchSubscription(const SDMComponent_ID& SubscriberId, const SDMComponent_ID&
source, const char* itemname, int ID, int& error, int debug);
49: bool removeSearchSubscription(const SDMComponent_ID& SubscriberId, int debug);
50: bool removeSearchSubscription(const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* itemname, int debug);
51: bool addVarReqSubscription(const SDMComponent_ID& SubscriberId, const SDMComponent_ID&
source, const char* interface, const char* itemname, int ID, int& error, int debug);
52: bool removeVarReqSubscription(const SDMComponent_ID& SubscriberId, int debug);
53: bool removeVarReqSubscription(const SDMComponent_ID& SubscriberId, const
SDMComponent_ID& source, const char* interface, const char* itemname, int debug);
54: SubscriptionList& operator=(const SubscriptionList&);
55: struct SubscriptionListNode* head;
56: struct SubscriptionListNode* tail;
57: };
58:
59: #endif

```

File: sdm/dm/xTEDSLibrary.h

```
1: #ifndef __SDM_XTEDS_LIBRARY_H_
2: #define __SDM_XTEDS_LIBRARY_H_
3:
4: #include "../common/xTEDS/xTEDS.h"
5: #include "../common/semaphore/semaphore.h"
6: #include "../common/message/SDMComponent_ID.h"
7: #include "../common/Regex/Regex.h"
8:
9:
10:
11: The xTEDSLibrary class is used by the DataManager to store a registered xTEDS for a device or
12: application, and all
13: needed information to handle the querying and subscribing to the device or application. This class
14: represents the
15: data nodes for a linked list (xTEDSLibraryList). When needed an xTEDSLibrary node is
16: dynamically allocated. If the
17: node is no longer needed, it is set to not being available but is not deallocated. The slot can be used
18: for a future
19: registration. This class also provides access to the tree representing the registered xTEDS, which is
20: constructed after
21: the xTEDS is set.
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:
869:
870:
871:
872:
873:
874:
875:
876:
877:
878:
879:
880:
881:
882:
883:
884:
885:
886:
887:
888:
889:
890:
891:
892:
893:
894:
895:
896:
897:
898:
899:
900:
901:
902:
903:
904:
905:
906:
907:
908:
909:
910:
911:
912:
913:
914:
915:
916:
917:
918:
919:
920:
921:
922:
923:
924:
925:
926:
927:
928:
929:
930:
931:
932:
933:
934:
935:
936:
937:
938:
939:
940:
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
953:
954:
955:
956:
957:
958:
959:
960:
961:
962:
963:
964:
965:
966:
967:
968:
969:
970:
971:
972:
973:
974:
975:
976:
977:
978:
979:
980:
981:
982:
983:
984:
985:
986:
987:
988:
989:
990:
991:
992:
993:
994:
995:
996:
997:
998:
999:
1000:
1001:
1002:
1003:
1004:
1005:
1006:
1007:
1008:
1009:
1010:
1011:
1012:
1013:
1014:
1015:
1016:
1017:
1018:
1019:
1020:
1021:
1022:
1023:
1024:
1025:
1026:
1027:
1028:
1029:
1030:
1031:
1032:
1033:
1034:
1035:
1036:
1037:
1038:
1039:
1040:
1041:
1042:
1043:
1044:
1045:
1046:
1047:
1048:
1049:
1050:
1051:
1052:
1053:
1054:
1055:
1056:
1057:
1058:
1059:
1060:
1061:
1062:
1063:
1064:
1065:
1066:
1067:
1068:
1069:
1070:
1071:
1072:
1073:
1074:
1075:
1076:
1077:
1078:
1079:
1080:
1081:
1082:
1083:
1084:
1085:
1086:
1087:
1088:
1089:
1090:
1091:
1092:
1093:
1094:
1095:
1096:
1097:
1098:
1099:
1100:
1101:
1102:
1103:
1104:
1105:
1106:
1107:
1108:
1109:
1110:
1111:
1112:
1113:
1114:
1115:
1116:
1117:
1118:
1119:
1120:
1121:
1122:
1123:
1124:
1125:
1126:
1127:
1128:
1129:
1130:
1131:
1132:
1133:
1134:
1135:
1136:
1137:
1138:
1139:
1140:
1141:
1142:
1143:
1144:
1145:
1146:
1147:
1148:
1149:
1150:
1151:
1152:
1153:
1154:
1155:
1156:
1157:
1158:
1159:
1160:
1161:
1162:
1163:
1164:
1165:
1166:
1167:
1168:
1169:
1170:
1171:
1172:
1173:
1174:
1175:
1176:
1177:
1178:
1179:
1180:
1181:
1182:
1183:
1184:
1185:
1186:
1187:
1188:
1189:
1190:
1191:
1192:
1193:
1194:
1195:
1196:
1197:
1198:
1199:
1200:
1201:
1202:
1203:
1204:
1205:
1206:
1207:
1208:
1209:
1210:
1211:
1212:
1213:
1214:
1215:
1216:
1217:
1218:
1219:
1220:
1221:
1222:
1223:
1224:
1225:
1226:
1227:
1228:
1229:
1230:
1231:
1232:
1233:
1234:
1235:
1236:
1237:
1238:
1239:
1240:
1241:
1242:
1243:
1244:
1245:
1246:
1247:
1248:
1249:
1250:
1251:
1252:
1253:
1254:
1255:
1256:
1257:
1258:
1259:
1260:
1261:
1262:
1263:
1264:
1265:
1266:
1267:
1268:
1269:
1270:
1271:
1272:
1273:
1274:
1275:
1276:
1277:
1278:
1279:
1280:
1281:
1282:
1283:
1284:
1285:
1286:
1287:
1288:
1289:
1290:
1291:
1292:
1293:
1294:
1295:
1296:
1297:
1298:
1299:
1300:
1301:
1302:
1303:
1304:
1305:
1306:
1307:
1308:
1309:
1310:
1311:
1312:
1313:
1314:
1315:
1316:
1317:
1318:
1319:
1320:
1321:
1322:
1323:
1324:
1325:
1326:
1327:
1328:
1329:
1330:
1331:
1332:
1333:
1334:
1335:
1336:
1337:
1338:
1339:
1340:
1341:
1342:
1343:
1344:
1345:
1346:
1347:
1348:
1349:
1350:
1351:
1352:
1353:
1354:
1355:
1356:
1357:
1358:
1359:
1360:
1361:
1362:
1363:
1364:
1365:
1366:
1367:
1368:
1369:
1370:
1371:
1372:
1373:
1374:
1375:
1376:
1377:
1378:
1379:
1380:
1381:
1382:
1383:
1384:
1385:
1386:
1387:
1388:
1389:
1390:
1391:
1392:
1393:
1394:
1395:
1396:
1397:
1398:
1399:
1400:
1401:
1402:
1403:
1404:
1405:
1406:
1407:
1408:
1409:
1410:
1411:
1412:
1413:
1414:
1415:
1416:
1417:
1418:
1419:
1420:
1421:
1422:
1423:
1424:
1425:
1426:
1427:
1428:
1429:
1430:
1431:
1432:
1433:
1434:
1435:
1436:
1437:
1438:
1439:
1440:
1441:
1442:
1443:
1444:
1445:
1446:
1447:
1448:
1449:
1450:
1451:
1452:
1453:
1454:
1455:
1456:
1457:
1458:
1459:
1460:
1461:
1462:
1463:
1464:
1465:
1466:
1467:
1468:
1469:
1470:
1471:
1472:
1473:
1474:
1475:
1476:
1477:
1478:
1479:
1480:
1481:
1482:
1483:
1484:
1485:
1486:
1487:
1488:
1489:
1490:
1491:
1492:
1493:
1494:
1495:
1496:
1497:
1498:
1499:
1500:
1501:
1502:
1503:
1504:
1505:
1506:
1507:
1508:
1509:
1510:
1511:
1512:
1513:
1514:
1515:
1516:
1517:
1518:
1519:
1520:
1521:
1522:
1523:
1524:
1525:
1526:
1527:
1528:
1529:
1530:
1531:
1532:
1533:
1534:
1535:
1536:
1537:
1538:
1539:
1540:
1541:
1542:
1543:
1544:
1545:
1546:
1547:
1548:
1549:
1550:
1551:
1552:
1553:
1554:
1555:
1556:
1557:
1558:
1559:
1560:
1561:
1562:
1563:
1564:
1565:
1566:
1567:
1568:
1569:
1570:
1571:
1572:
1573:
1574:
1575:
1576:
1577:
1578:
1579:
1580:
1581:
1582:
1583:
1584:
1585:
1586:
1587:
1588:
1589:
1590:
1591:
1592:
1593:
1594:
1595:
1596:
1597:
1598:
1599:
1600:
1601:
1602:
1603:
1604:
1605:
1606:
1607:
1608:
1609:
1610:
1611:
1612:
1613:
1614:
1615:
1616:
1617:
1618:
1619:
1620:
1621:
1622:
1623:
1624:
1625:
1626:
1627:
1628:
1629:
1630:
1631:
1632:
1633:
1634:
1635:
1636:
1637:
1638:
1639:
1640:
1641:
1642:
1643:
1644:
1645:
1646:
1647:
1648:
1649:
1650:
1651:
1652:
1653:
1654:
1655:
1656:
1657:
1658:
1659:
1660:
1661:
1662:
1663:
1664:
1665:
1666:
1667:
1668:
1669:
1670:
1671:
1672:
1673:
1674:
1675:
1676:
1677:
1678:
1679:
1680:
1681:
1682:
1683:
1684:
1685:
1686:
1687:
1688:
1689:
1690:
1691:
1692:
1693:
1694:
1695:
1696:
1697:
1698:
1699:
1700:
1701:
1702:
1703:
1704:
1705:
1706:
1707:
1708:
1709:
1710:
1711:
1712:
1713:
1714:
1715:
1716:
1717:
1718:
1719:
1720:
1721:
1722:
1723:
1724:
1725:
1726:
1727:
1728:
1729:
1730:
1731:
1732:
1733:
1734:
1735:
1736:
1737:
1738:
1739:
1740:
1741:
1742:
1743:
1744:
1745:
1746:
1747:
1748:
1749:
1750:
1751:
1752:
1753:
1754:
1755:
1756:
1757:
1758:
1759:
1760:
1761:
1762:
1763:
1764:
1765:
1766:
1767:
1768:
1769:
1770:
1771:
1772:
1773:
1774:
1775:
1776:
1777:
1778:
1779:
1780:
1781:
1782:
1783:
1784:
1785:
1786:
1787:
1788:
1789:
1790:
1791:
1792:
1793:
1794:
1795:
1796:
1797:
1798:
1799:
1800:
1801:
1802:
1803:
1804:
1805:
1806:
1807:
1808:
1809:
1810:
1811:
1812:
1813:
1814:
1815:
1816:
1817:
1818:
1819:
1820:
1821:
1822:
1823:
1824:
1825:
1826:
1827:
1828:
1829:
1830:
1831:
1832:
1833:
1834:
1835:
1836:
1837:
1838:
1839:
1840:
1841:
1842:
1843:
1844:
1845:
1846:
1847:
1848:
1849:
1850:
1851:
1852:
1853:
1854:
1855:
1856:
1857:
1858:
1859:
1860:
1861:
1862:
1863:
1864:
1865:
1866:
1867:
1868:
1869:
1870:
1871:
1872:
1873:
1874:
1875:
1876:
1877:
1878:
1879:
1880:
1881:
1882:
1883:
1884:
1885:
1886:
1887:
1888:
1889:
1890:
1891:
1892:
1893:
1894:
1895:
1896:
1897:
1898:
1899:
1900:
1901:
1902:
1903:
1904:
1905:
1906:
1907:
1908:
1909:
1910:
1911:
1912:
1913:
1914:
1915:
1916:
1917:
1918:
1919:
1920:
1921:
1922:
1923:
1924:
1925:
1926:
1927:
1928:
1929:
1930:
1931:
1932:
1933:
1934:
1935:
1936:
1937:
1938:
1939:
1940:
1941:
1942:
1943:
1944:
1945:
1946:
1947:
1948:
1949:
1950:
1951:
1952:
1953:
1954:
1955:
1956:
1957:
1958:
1959:
1960:
1961:
1962:
1963:
1964:
1965:
1966:
1967:
1968:
1969:
1970:
1971:
1972:
1973:
1974:
1975:
1976:
1977:
1978:
1979:
1980:
1981:
1982:
1983:
1984:
1985:
1986:
1987:
1988:
1989:
1990:
1991:
1992:
1993:
1994:
1995:
1996:
1997:
1998:
1999:
2000:
2001:
2002:
2003:
2004:
2005:
2006:
2007:
2008:
2009:
2010:
2011:
2012:
2013:
2014:
2015:
2016:
2017:
2018:
2019:
2020:
2021:
2022:
2023:
2024:
2025:
2026:
2027:
2028:
2029:
2030:
2031:
2032:
2033:
2034:
2035:
2036:
2037:
2038:
2039:
2040:
2041:
2042:
2043:
2044:
2045:
2046:
2047:
2048:
2049:
2050:
2051:
2052:
2053:
2054:
2055:
2056:
2057:
2058:
2059:
2060:
2061:
2062:
2063:
2064:
2065:
2066:
2067:
2068:
2069:
2070:
2071:
2072:
2073:
2074:
2075:
2076:
2077:
2078:
2079:
2080:
2081:
2082:
2083:
2084:
2085:
2086:
2087:
2088:
2089:
2090:
2091:
2092:
2093:
2094:
2095:
2096:
2097:
2098:
2099:
2100:
2101:
2102:
2103:
2104:
2105:
2106:
2107:
2108:
2109:
2110:
2111:
2112:
2113:
2114:
2115:
2116:
2117:
2118:
2119:
2120:
2121:
2122:
2123:
2124:
2125:
2126:
2127:
2128:
2129:
2130:
2131:
2132:
2133:
2134:
2135:
2136:
2137:
2138:
2139:
2140:
2141:
2142:
2143:
2144:
2145:
2146:
2147:
2148:
2149:
2150:
2151:
2152:
2153:
2154:
2155:
2156:
2157:
2158:
2159:
2160:
2161:
2162:
2163:
2164:
2165:
2166:
2167:
2168:
2169:
2170:
2171:
2172:
2173:
2174:
2175:
2176:
2177:
2178:
2179:
2180:
2181:
2182:
2183:
2184:
2185:
2186:
2187:
2188:
2189:
2190:
2191:
2192:
2193:
2194:
2195:
2196:
2197:
2198:
2199:
2200:
2201:
2202:
2203:
2204
```

```

33: void setTargetPort(unsigned short new_targetport);
34: void setConnections(int new_connections);
35: void setActive(bool new_active);
36: void setPid(unsigned long new_pid);
37: void setHub(int new_hub);
38: void setSPANode(char* new_locationAddress);
39: void setPosted(bool new_posted);
40: void setComponentID(SDMComponent_ID new_id);
41: void setSPAHub(char* new_spahub);
42: void setMerged(bool new_merged);
43: const char* getName(void);
44: const char* getXTEDS(void);
45: unsigned long getAddress(void);
46: unsigned long getSensorID(void);
47: bool getAvailable(void);
48: unsigned short getTargetPort(void);
49: int getConnections(void);
50: bool getActive(void);
51: unsigned long getPid(void);
52: int getHub(void);
53: char* getSPANode(void) { return m_strSPANode; }
54: const char* getSPAHub(void);
55: const char* getSPAPort(void);
56: bool getPosted(void);
57: const char* getComponentKey(void);
58: bool getMerged(void);
59: SDMComponent_ID getComponentID(void);
60: RegexResult XtedsRegexSearchCapturesOnly(const RegularExpression& Pattern);
61:
62: xTEDSLibrary& operator=(const xTEDSLibrary&);
63:
64: Sem* inUse;           //Semaphore to provide mutual exclusion, all reading threads must lock
before accessing
65: xTEDS* xtedsTree;     //The tree structure storing the xTEDS information
66:
67: private:
68: bool available;       //Whether this xTEDSLibrary slot is being used
69: int connections;      //The number of subscribed connections to the xTEDS (number of messages
consumed)
70: bool active;          //Whether the device or application is currently active
71: unsigned long pid;     //The PID of the xTEDS

```

```
72: int hub;           //The hub number of the device
73: bool posted;       //Whether a task is posted to the TaskManager
74: char* m_strSPANode; //The SPA or USB address of a device
75: char* fullxTED;    //The xTEDS of the device or application
76: char* tempSPAHub;  //The SPAHub address
77: SDMComponent_ID* id; //The component identifier of the xTEDS
78: bool merged;       //The status of whether the xTEDS has been merged
79: };
80:
81: #endif
```

File: sdm/dm/DM.h

```
1: #ifndef __SDM_DM_H_
2: #define __SDM_DM_H_
3:
4: #include "../common/UDPcom.h"
5: #include "../common/TCPcom.h"
6: #include "../common/message_defs.h"
7: #include "../common/version.h"
8: #include "../common/message/SDMSubreqst.h"
9: #include "../common/message/SDMDeletesub.h"
10: #include "../common/message/SDMConsume.h"
11: #include "../common/message/SDMReqReg.h"
12: #include "../common/message/SDMRegInfo.h"
13: #include "../common/message/SDMCancel.h"
14: #include "../common/message/SDMService.h"
15: #include "../common/message/SDMSerreqst.h"
16: #include "../common/message/SDMCommand.h"
17: #include "../common/message/SDMReqxTEDS.h"
18: #include "../common/message/SDMxTEDSInfo.h"
19: #include "../common/message/SDMmessage.h"
20: #include "../common/message/SDMCancelxTEDS.h"
21: #include "../common/message/SDMxTEDS.h"
22: #include "../common/message/SDMTat.h"
23: #include "../common/message/SDMPostTask.h"
24: #include "../common/message/SDMData.h"
25: #include "../common/message/SDMReady.h"
26: #include "../common/message/SDMSearch.h"
27: #include "../common/message/SDMSearchReply.h"
28: #include "../common/message/SDMElection.h"
29: #include "../common/message/SDMDMLLeader.h"
30: #include "../common/message/SDMAck.h"
31: #include "../common/message/SDMVarReq.h"
32: #include "../common/message/SDMTaskError.h"
33: #include "../common/message/SDMVarInfo.h"
34: #include "../common/message/SDMHeartbeat.h"
35: #include "../common/message/SDMHello.h"
36: #include "../common/message/SDMRegister.h"
37: #include "../common/message/SDMID.h"
38: #include "../common/MessageLogger/MessageLogger.h"
39: #include "../common/MessageManager/MessageManager.h"
```



```

40: #include "../common/Debug.h"
41: #include "../common/Exception/SDMRegexException.h"
42: #include "../common/SDMRegQueue.h"
43: #include "../common/SDMCancelQueue.h"
44: #include "Parse.h"
45: #include "xTEDSLibraryList.h"
46: #include "xTEDSParameters.h"
47: #include "SubscriptionList.h"
48: #include "backupDMList.h"
49: #include "xTEDSSegmentBuilder.h"
50: #include "DMUtils.h"
51: #include "ProviderSubscriptionList.h"
52: //
53: //Data Manager xTEDS message identifiers
54: //
55: const SDMMMessage_ID INVALID_MESSAGE_ID(0, 0);
56: //DM Notification messages
57: const SDMMMessage_ID NOTI_REGISTRATION(1, 1);
58: const SDMMMessage_ID NOTI_DEREGISTRATION(1, 2);
59: const SDMMMessage_ID NOTI_XTEDS_MODIFICATION(1, 3);
60: const SDMMMessage_ID NOTI_REGISTRATION_CHANGE(1, 4);
61: const SDMMMessage_ID NOTI_PERFORMANCE_COUNTERS(2, 13);
62:
63: //Data reply messages
64: const SDMMMessage_ID RPLY_CONVERTED_DEVICE_NAME(1, 5);
65: const SDMMMessage_ID RPLY_CONVERTED_SPANODE(1, 7);
66: const SDMMMessage_ID RPLY_CONVERTED_IP(1, 10);
67: const SDMMMessage_ID RPLY_RETURN_COMP_KEY(1, 15);
68: const SDMMMessage_ID RPLY_ERRORS(2, 19);
69:
70: //Command requests
71: const SDMMMessage_ID CMD_ENABLE_LOGGING(3, 16);
72: const SDMMMessage_ID CMD_DISABLE_LOGGING(3, 17);
73:
74: //Service requests
75: const SDMMMessage_ID SER_SEND_SENSOR_ID(1, 6);
76: const SDMMMessage_ID SER_SEND_PID(1, 8);
77: const SDMMMessage_ID SER_SID_TO_SPANODE(1, 11);
78: const SDMMMessage_ID SER_SID_TO_IP(1, 12);
79: const SDMMMessage_ID SER_COMPID_TO_COMPKEY(1, 14);
80: const SDMMMessage_ID SER_GET_ERRORS(2, 18);

```

```

81:
82: //xTEDS modification types
83: #define MOD_NOT_APPLICABLE    0x0
84: #define REGISTRATION_MODIFICATION    0x1
85: #define DEREGISTRATION_MODIFICATION    0x2
86: #define UPDATE_MODIFICATION    0x3
87:
88: //xTEDS Hub types
89: #define APPLICATION    0x0
90: #define DEVICE    0x1
91: #define ROBOHUB    0x2
92:
93: //
94: //Data Manager Function Prototypes
95: //
96: void* RegistrationHandler(void* args);
97: void* CancelRegHandler(void* args);
98: double GetCurTime();
99: void TaskError(char *buf, int size, const SDMComponent_ID& RequesterId);
100: void Consume(char *buf, int size, const SDMComponent_ID& RequesterId);
101: void Cancel(char *buf, int size, const SDMComponent_ID& RequesterId);
102: void Command(char *buf, int size, const SDMComponent_ID& SenderId);
103: void Service(char *buf, int size, const SDMComponent_ID& RequesterId);
104: void Serreqst(const char* buf, int size, const SDMComponent_ID& RequesterId);
105: void ReqReg(char *buf, int size, const int xTEDref, const SDMComponent_ID& RequesterId, int
xTEDSAction);
106: void Search(char *buf, int size, const SDMComponent_ID& RequesterId, int xTEDref);
107: int SendAckMessage(short sAckStatus, bool bUseComponentId, const SDMComponent_ID&
idDest, const SDMComHandle& AppHandle, bool bForceSend = false);
108: void* xTEDS(void* arg);//char *buf, int size, char *ipaddr
109: void* CancelxTEDS(void* arg);//char *buf, int size, char *ipaddr);
110: void ReqxTEDS(char *buf, int size, const SDMComponent_ID& RequesterId);
111: void Subscribe( const SDMComponent_ID& RequesterId, const SDMMMessage_ID& mID);
112: void CancelSubscription(const SDMComponent_ID& RequesterId, int mID);
113: void PublishxTEDSModificationSubscription(unsigned long SensorID, int MessageID, int
ModificationAction, int xTEDSRef);
114: void PublishNotification(unsigned long SensorID, SDMMMessage_ID MessageID, int
ModificationAction = MOD_NOT_APPLICABLE);
115: void Publish(SubscriptionListNode* node, unsigned long sID, int mID, int xTEDref);
116: void Ready(char *buf, int size, const SDMComponent_ID& RequesterId);
117: void ServicePublish(const SDMComponent_ID& RequesterId, unsigned int sID, const
SDMMMessage_ID& DataReplyId, int seq_num);

```

```

118: void TAT(char *buf, int size, const SDMComponent_ID& RequesterId);
119: void PipeInit();
120: void* ChildFunctionCallProcess(void*);
121: void* UdpListenerProcess(void*);
122: void* TcpListenerProcess(void*);
123: void PrintxTEDS();
124: xTEDSLibraryListNode* AlreadyRegistered(char* sensor, const char* buf, unsigned short port,
SDMComponent_ID& idAddr, int type, int& spot);
125: xTEDSLibraryListNode* StoreInfo(char* sensor, int& spot, const SDMComponent_ID& idAddr,
const char* buf, int type);
126: void PostTask(xTEDSLibraryListNode* node);
127: void MessageSent(SDMmessage *msg);
128: void MessageReceived(SDMmessage *msg);
129: xTEDSLibraryListNode* MatchSID(long sID);
130: void* UpdatexTEDS(void* arg);
131: void* SendxTEDS(void* arg);
132: void MergexTEDS(xTEDSLibraryListNode* flag, bool update, SDMxTEDS& ted, int xTEDref);
133: bool MergexTEDS(xTEDSLibraryListNode* flag, bool update, SDMxTEDS& ted, int xTEDref, int
Hubvalue);
134: void VarReq(char* buf, int size, const SDMComponent_ID& RequesterId, int xTEDref);
135: bool IAmElected();
136: #ifndef WIN32
137: void* SigHandler(void*);
138: #else
139: void SigHandler(int);
140: #endif //WIN32
141:
142:
143: #ifdef PNP_BACKUP
144: void NMElection(MessageManager* mm);
145: bool AmIPrimary(MessageManager* mm);
146: void* Heartbeat(void*);
147: void QueueHeartbeat (char buf[], long length);
148: void NonElected(MessageManager* mm);
149: int RunBackupListener(void);
150: void BackupModifyReqRegSubscription(const char* msgBuf);
151: void BackupModifySearchSubscription(const char* msgBuf);
152: void BackupModifyVarReqSubscription(const char* msgBuf);
153: void BackupConsume(const char* msgBuf);
154: void BackupTaskError(const char* msgBuf);
155: #ifdef PNP_FAKE
156: void DMSendIMA();

```

```
157: #endif    //PNP_FAKE
158: #endif    //PNP_BACKUP
159:
160:
161:
162: #define xsize 25
163:
164: #endif //->#ifndef __SDM_DM_H_
```

File: sdm/dm/Subscription.h

```
1:
/*****
*****
2:
3: The Subscription class holds the necessary information needed to create a subscription entry for the
SubscriptionList
4: class. These subscription entries are used by the DataManager for various device and application
subscriptions.
5: Including subscription to the DataManager's notification messages, subscriptions to SDMReqReg
queries with future and
6: cancellation reply types, and subscriptions to SDMSearch queries with future reply types.
7:
8:
*****/
9: #ifndef __SDM_SUBSCRIPTION_H_
10: #define __SDM_SUBSCRIPTION_H_
11:
12: #include "../common/message/SDMComponent_ID.h"
13: #include "../common/message/SDMMessage_ID.h"
14:
15: class Subscription
16: {
17: public:
18: Subscription();
19: Subscription(const Subscription&);
20:
21: virtual ~Subscription();
22:
23: void setAddress(char* new_IP);
24: void setPort(unsigned short new_port);
25: void setmID(int new_mID); SDM_DEPRECATED
26: void setmID(const SDMMessage_ID& NewMid);
27: void setInuse(bool new_inuse);
28: void setID(int new_ID);
29: void setItemName(const char* new_itemname);
30: void setQuallist(const char* new_quallist);
31: void setItems(bool new_items);
32: void setDestination(const SDMComponent_ID& new_destination);
33: void setSource(const SDMComponent_ID& new_source);
```

```

34: void setDevice(const char* new_device);
35: void setInterface(const char* new_interface);
36: char* getAddress(void) const;
37: unsigned short getPort(void) const;
38: int getmID(void) const;
39: bool getInuse(void) const { return m_bInUse; }
40: const char* getItemName(void) const { return m_strItemName; }
41: const char* getQualList(void) const { return m_strQualList; }
42: bool getItems(void) const { return m_bHasItems; }
43: int getID(void) const { return m_iID; }
44: SDMComponent_ID getDestination(void) const { return m_idSubscriber; }
45: SDMComponent_ID getSource(void) const { return m_idSource; }
46: const char* getDevice(void) const { return m_strDevice; }
47: const char* getInterface(void) const { return m_strInterface; }
48:
49: Subscription& operator=(const Subscription&);
50: private:
51: SDMMMessage_ID m_SubMessageID; //The message ID to which this entry is subscribed
52: bool m_bInUse; //Whether the subscription entry is in use
53: int m_iID; //The ID number associated with this subscription stream
54: char* m_strItemName; //The item_name field for the subscription
55: char* m_strQualList; //The qual_list field for the subscription
56: SDMComponent_ID m_idSubscriber; //The destination component identifier for subscription
replies
57: SDMComponent_ID m_idSource; //The source component identifier of the interested device
58: char* m_strDevice; //The device field for the subscription
59: char* m_strInterface; //The interface field for the subscription
60: bool m_bHasItems; //Whether this list is associated with subscription items
61: };
62:
63: #endif

```

File: sdm/dm/xTEDSPParameters.h

```
1: #ifndef __XTEDS_PARAMETERS_H_
2: #define __XTEDS_PARAMETERS_H_
3:
4: #include "../common/SDMComHandle.h"
5: #include "../common/message/SDMComponent_ID.h"
6:
7:
8:
9: The xTEDSPParameters class is used to pass data to various xTEDS handling threads and
10: functions within the DataManager. It contains the information needed for the address,
11: socket, and buffers containing the needed data.
12:
13:
14:
15: class xTEDSPParameters
16: {
17: public:
18: xTEDSPParameters();
19: xTEDSPParameters(const xTEDSPParameters&);
20: xTEDSPParameters(char* buffer,int length,const SDMComponent_ID& Sender, const
SDMComHandle& ComHandle);
21: ~xTEDSPParameters();
22: xTEDSPParameters& operator=(const xTEDSPParameters&);
23:
24: const char* getBuffer() const { return m_MessageData; }
25: int getSize() const { return m_iDataSize; }
26: const SDMComponent_ID& getSenderAddress() const { return m_idSender; }
27:
28: const SDMComHandle& GetComHandle() const { return m_comHandle; }
29: void ComCleanup() { m_comHandle.DoCleanup(); }
30: private:
31: char* m_MessageData; //Buffer containing the data to pass
32: int m_iDataSize; //Size of the buffer
33: SDMComponent_ID m_idSender;
34: SDMComHandle m_comHandle;
35: };
```

36:
37: #endif

File: sdm/dm/xTEDSSegmentBuilder.h

```
1: /*
2: * The xTEDSSegmentBuilder class is used by the SDM Data Manager to receive xTEDS documents
3: * from connected sensors who do not have the capability for TCP/IP communication, and have
4: * an xTEDS to publish that is larger than the size of a UDP datagram. Multiple transmissions
5: * can occur simultaneously, however mutual exclusion is assumed to be handled at the layer
6: * above this class. Each transmission of xTEDS is differentiated based on the sensor's
7: * SDMComponent_ID.
8: */
9:
10: #ifndef _XTEDS_SEGMENT_BUILDER_H_
11: #define _XTEDS_SEGMENT_BUILDER_H_
12: #include "../common/message/SDMComponent_ID.h"
13: #include "../common/message/SDMxTEDS.h"
14:
15: #define MAX_NUMBER_SEGMENTS 4
16: #define SEGMENT_MAX_XTEDS_SIZE 8070
17: #define MAX_SEGMENT_NODES 5
18:
19: //Structure representing an xTEDS to build, as messages come in
20: struct xTEDSSegmentNode
21: {
22:     bool IsInUse;
23:     char xTEDSBuffer[MAX_NUMBER_SEGMENTS*SEGMENT_MAX_XTEDS_SIZE]; //Pointer to
the xTEDS being built
24:     unsigned short NumSegments; //The number of xTEDS segments to receive
25:     unsigned char SegmentsReceived[16]; //An array of which segments have been received
26:     SDMComponent_ID xTEDSID; //Component identifier of the xTEDS provider
27:     int NextIndex; //Index position to the next node in the list
28: //Initializer
29: xTEDSSegmentNode() : IsInUse(false),NumSegments(0),xTEDSID(),NextIndex(-1) {}
30: //Declared but not defined:
31: xTEDSSegmentNode(const xTEDSSegmentNode &right);
32: xTEDSSegmentNode &operator=(const xTEDSSegmentNode &right);
33: };
34:
35: class xTEDSSegmentBuilder
36: {
37: public:
38: xTEDSSegmentBuilder();
```

```

39: ~xTEDSSegmentBuilder();
40: xTEDSSegmentBuilder(const xTEDSSegmentBuilder &right);
41: xTEDSSegmentBuilder& operator=(const xTEDSSegmentBuilder &right);
42:
43: bool ApplySegment(const SDMxTEDS &Message);
44: bool CheckIsFinished(const SDMxTEDS &Message);
45: bool GetFullxTEDS(const SDMxTEDS &Message, char *xTEDSOut, int MaxSize);
46: static bool IsSegmentedxTEDS(const SDMxTEDS &Message);
47: private:
48: xTEDSSegmentNode NodeList[MAX_SEGMENT_NODES];
49: xTEDSSegmentNode* AddSegmentNode();
50: bool DeleteNode(const SDMComponent_ID &xTEDSID);
51: xTEDSSegmentNode* FindNodeEntry(const SDMComponent_ID &xTEDSID);
52: bool SentInOrder(unsigned char SequenceNumber, xTEDSSegmentNode* CurrNode);
53: bool AlreadyReceived(unsigned char SequenceNumber, xTEDSSegmentNode* CurrNode);
54: };
55:
56: #endif

```

File: sdm/dm/Parse.h

```
1: //Parse.h
2:
3: //ReqReg return types
4: #define REQTYPE_REGEX      2
5: #define REQTYPE_EMPTYITEM  1
6: #define REQTYPE_USEITEM    0
7:
8: #define SDM_CONFIG_FILE_NAME  "sdm.config"
9:
10: //Function prototypes
11: int ParseDeviceName(char *buf, char *sensor_buf, unsigned int sensor_buflen, int level);
12: int GetAttribute(const char *item_def, const char *attribute, char *attribute_value, unsigned int attribute_valuelen, int level);
13: int ParseItemName(char *item, int level);
14: int MsgIDFrom(char *buf, int mID, int level);
15: int MergeConfigxTED(char *xTED, int xTEDBufLength, char *config, int level);
16: int FindConfigInfo(char *type, char* info, char* hub, char* port, int level);
17: //int GetSpaHubInfo(char *buf, char *info, int level);
18: int GetSPAHub(char* buf, char* hub, int level);
19: bool FindDevicesHubPath (const char *DevicePath, char *HubPathOut, unsigned int HubPathOutLen, int DebugLevel);
```

File: sdm/dm/ProviderSubscription.h

```
1: #ifndef _SDM_PROVIDER_SUBSCRIPTION_H_
2: #define _SDM_PROVIDER_SUBSCRIPTION_H_
3:
4: #include "../common/message/SDMComponent_ID.h"
5: #include "../common/message/SDMMessage_ID.h"
6:
7: class ProviderSubscription
8: {
9: public:
10: ProviderSubscription();
11: void Set(const SDMComponent_ID& idProvider, const SDMComponent_ID& idSubscriber, const
SDMMessage_ID& idMessage);
12:
13: const SDMComponent_ID& GetSubscriber() const { return m_SubscriberId; }
14: const SDMComponent_ID& GetProvider() const { return m_ProviderId; }
15: const SDMMessage_ID& GetMessage() const { return m_MessageId; }
16: private:
17: SDMComponent_ID m_ProviderId;
18: SDMComponent_ID m_SubscriberId;
19: SDMMessage_ID m_MessageId;
20: };
21:
22: #endif
```

Listing from directory: sdm/pm

File: sdm/pm/pm.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <stdlib.h>
4: #include <sys/wait.h>
5: #include <sys/stat.h>
6: #include <errno.h>
7: #include <unistd.h>
8: #include <ctype.h>
9: #ifndef __VXWORKS__
10: #include <getopt.h>
11: #include <sys/poll.h>
12: #else
13: #include <iolib.h>
14: #include <selectLib.h>
15: #endif
16: #include <sys/socket.h>
17: #include <sys/types.h>
18: #include <sys/time.h>
19: #include <netinet/in.h>
20: #include <arpa/inet.h>
21: #include <vector>
22: #include "../common/message/SDMCancel.h"
23: #include "../common/message/SDMReqReg.h"
24: #include "../common/message/SDMRegInfo.h"
25: #include "../common/message/SDMConsume.h"
26: #include "../common/message/SDMData.h"
27: #include "../common/message/SDMCode.h"
28: #include "../common/message/SDMReqCode.h"
29: #include "../common/semaphore/semaphore.h"
30: #include "../common/Debug.h"
31: #include "pm.h"
32: #include "pm_main.h"
33:
34: using namespace std;
35:
36: extern SDMMessage_ID tm_mode_msg_id;
37: extern unsigned long Address_PM;
```

```

38: extern SDMComponent_ID tm_mode_source;
39: extern int debug;
40: extern bool bgTMDetected;
41: extern unsigned char tm_mode;
42: extern vector<code_rcv> codeReceivers;
43: extern pthread_mutex_t codeReceiversMutex;
44: extern Sem g_semLaunchTask;
45:
46: /*
47:  * Cancel the PM's registration subscriptions with the Data Manager.
48:  */
49: void CancelReqRegSubs(void)
50: {
51:   SDMCancel msgCancel;
52:   // Cancel the subscription to the TaskManager's mode message
53:   msgCancel.msg_id = tm_mode_msg_id;
54:   msgCancel.source = tm_mode_source;
55:   msgCancel.destination.setPort(PORT_PM);
56:   msgCancel.destination.setAddress(Address_PM);
57:
58:   msgCancel.Send();
59:   MessageSent(&msgCancel);
60: }
61:
62: /*
63: ReqRegMode puts together and sends an SDMReqReg message for the TaskManager's mode
message. The mode message
64: will be used to reset the ProcessManager if the DataManager has failed.
65: INPUTS:
66:     None.
67: Returns:
68:     None.
69: */
70: void ReqRegSubs(void)
71: {
72:   SDMReqReg msgReqReg;
73:   // SDMReqReg for TM's status message
74:   debug_f(2,"Sending ReqReg for TM's status notification. \n");
75:   msgReqReg.reply = SDM_REQREG_CURRENT_AND_FUTURE;
76:   msgReqReg.id = TM_Mode_ID;
77:   strcpy(msgReqReg.device, "TaskManager");

```

```

78: strcpy(msgReqReg.interface, "TM_Interface");
79: strcpy(msgReqReg.item_name, "Status");
80: msgReqReg.destination.setAddress(Address_PM);
81: msgReqReg.destination.setPort(PORT_PM);
82:
83: msgReqReg.Send();
84: MessageSent(&msgReqReg);
85: }
86:
87:
88: /*
89: RegInfoHandler handles the reception of SDMRegInfo messages. The ProcessManager expects to
receive an SDMRegInfo
90: message for the TaskManager's mode message which will be used to reset the PM node upon failure
of the DataManager.
91: INPUTS:
92:     buf - The buffer containing the SDMRegInfo message.
93: RETURNS:
94:     None.
95: */
96: void RegInfoHandler (const char *buf)
97: {
98:     SDMRegInfo msgRegInfo;
99:     SDMConsume msgConsume;
100:     long lResult;
101:     if ((lResult=msgRegInfo.Unmarshal(buf)) == SDM_INVALID_MESSAGE)
102:     {
103:         printf("Invalid SDMRegInfo message received. \n");
104:         return;
105:     }
106:     MessageReceived(&msgRegInfo);
107:     if (lResult == SDM_NO_FURTHER_DATA_PROVIDER)
108:         return;
109:
110:     if (msgRegInfo.id == TM_Mode_ID)
111:     {
112:         debug_f(2,"TM status notification found, sending subscribe request. \n");
113:         msgConsume.source = msgRegInfo.source;
114:         msgConsume.msg_id = msgRegInfo.msg_id;
115:         tm_mode_msg_id = msgRegInfo.msg_id.getInterfaceMessagePair();
116:         tm_mode_source = msgRegInfo.source;

```

```

117:     msgConsume.destination.setPort(PORT_PM);
118:     msgConsume.Send();
119:     MessageSent(&msgConsume);
120:     /*If an SDMRegInfo is received, as a result of the TM restarting after a fault, reregister with
the TM */
121:     if (bgTMDetected) /*If this is the first series of RegInfo's received, don't reset or re-register
(bgTMDetected is false) */
122:     {
123:         //DoReset();
124:         SendRunningTasks();
125:         RegisterPM();
126:     }
127:     else
128:         bgTMDetected = true; /*The next RegInfo's received will be because of a fault */
129: }
130: }
131:
132:
133: /*
134: DataHandler handles the reception of SDMDData messages. The ProcessManager expects to
received an SDMDData message
135: containing the TaskManager's current mode. The mode is used to reset the PM node upon failure
of the DataManager.
136: INPUTS:
137:     buf - The buffer containing the SDMDData message.
138: RETURNS:
139:     None.
140: */
141: void DataHandler (const char *buf)
142: {
143:     SDMDData msgData;
144:     SDMComponent_ID newDmId;
145:     unsigned char ucOldMode;
146:     if (msgData.Unmarshal(buf) < 0)
147:     {
148:         printf("Invalid SDMDData message. \n");
149:         return;
150:     }
151:     MessageReceived(&msgData);
152:
153:     // If this SDMDData message is the TaskManager's mode notification
154:     if (msgData.msg_id == tm_mode_msg_id)

```



```

155:  {
156:      // Hang on to the current mode
157:      ucOldMode = tm_mode;
158:      tm_mode = msgData.msg[0];
159:
160:      debug_f(1,"TM Mode change notification received, new mode is %hhd. \n",tm_mode);
161:      if (tm_mode == MODE_HARD_RESET)
162:      {
163:          debug_f(1, "   Performing a hard reset... \n");
164:          // The DataManager has failed, get the new DM's address from the message
165:          newDmId.Unmarshal(&msgData.msg[1], 0);
166:          DataManager = newDmId;
167:
168:          debug_f(1,"New DM address is 0x%lx:%hd %ld \n", newDmId.getAddress(),
newDmId.getPort(), newDmId.getSensorID());
169:          // Perform a reset
170:          DoReset();
171:          // For a hard reset, reregister the PM's xTEDS
172:          RegisterXteds();
173:      }
174:      else if (tm_mode == MODE_SOFT_RESET)
175:      {
176:          debug_f(1,"   Performing a soft reset... \n");
177:          // Perform the reset
178:          DoReset();
179:      }
180:      // Re-set the old mode
181:      tm_mode = ucOldMode;
182:  }
183: }
184:
185: /*
186:  HandleCodeMessage handles an incoming code message, determines which child process (if
multiples) is waiting for this message
187:  , and sends the message to the child process through a pipe that was previously set up.
188:  INPUTS:
189:      buf - The buffer containing the SDMCode message.
190:  RETURNS:
191:      None.
192: */
193: void HandleCodeMessage(const char *buf)

```

```

194: {
195:     SDMCode msgCode;
196:     unsigned int receiver_index;
197:     if (msgCode.Unmarshal(buf) < 0)
198:     {
199:         printf("Invalid SDMCode message. \n");
200:         return;
201:     }
202:     MessageReceived(&msgCode);
203:     debug_f(1,"Code message received. \n");
204:     pthread_mutex_lock(&codeReceiversMutex);
205:     // Find which codeReceiver matches the data message
206:     for (receiver_index = 0; receiver_index < codeReceivers.size(); receiver_index++)
207:     {
208:         if (!strcmp(codeReceivers[receiver_index].filename, msgCode.filename))
209:             break;
210:     }
211:
212:     // If the end of the vector was reached before finding the match, an error occurred
213:     if (receiver_index == codeReceivers.size())
214:     {
215:         debug_f(2,"SDMCode message received which does not match any code-consuming process.
\n");
216:         pthread_mutex_unlock(&codeReceiversMutex);
217:         return;
218:     }
219:     // Write the code message to the pipe
220:     else if (write(codeReceivers[receiver_index].fd_code, buf, BUFSIZE) <= 0)
221:     {
222:         perror("Error writing code message to code pipe in HandleCodeMessage(). \n");
223:     }
224:     pthread_mutex_unlock(&codeReceiversMutex);
225: }
226:
227: /*
228:  The GetCode routine requests executable code from the TM
229:  INPUTS:
230:      filename - name of the code desired
231:      fd_readcode - file descriptor for the pipe to read the code message
232:      fd_writestatus - file descriptor for the pipe to write the status message for the transfer
233:  RETURN VALUE:

```

```

234:         true - code successfully retrieved
235:         false - code not retrieved
236: */
237: bool GetCode (const char* strFilename, long pid, int iVersion, int iFdReadCode)
238: {
239:     char msgBuffer[BUFSIZE];
240:     int iLength;
241:     bool bFinished = false, bValid = true;
242:     SDMCode msgCode;
243:     unsigned short usSequenceNumber, usNumSegments;
244:     const int MAX_CODE_SIZE = sizeof(msgCode.code);
245:     unsigned short usCurSequenceNumber = 1, usPollMiss = 0;
246:
247: #ifdef __VXWORKS__
248:     char fullFilename[MAX_FILENAME_SIZE];
249:     strcpy(fullFilename, "/ram0/");
250:     strcat(fullFilename, strFilename);
251:     strcat(fullFilename, ".vxe");
252:     g_semLaunchTask.Wait();
253:     FILE* pFile = fopen(fullFilename, "wb");
254:     if(pFile == NULL)
255:     {
256:         printf("ERROR creating file %s for code transfer: %s \n", fullFilename, strerror( errno ));
257:     }
258: #else
259:     g_semLaunchTask.Wait();
260:     FILE* pFile = fopen(strFilename, "wb");
261:     if(pFile == NULL)
262:     {
263:         printf("ERROR creating file %s for code transfer: %s \n", strFilename, strerror( errno ));
264:     }
265: #endif
266:
267: #ifndef __VXWORKS__
268:     if (SetCloseFileOnExec(fileno(pFile)) == -1)
269:         printf("Error setting new file to close on fork. \n");
270: #endif
271:     g_semLaunchTask.Signal();
272:
273:     SDMReqCode msgRequestCode;
274:     strncpy(msgRequestCode.filename, strFilename, sizeof(msgRequestCode.filename));

```

```

275:  msgRequestCode.source.setAddress(Address_PM);
276:  msgRequestCode.source.setPort(PORT_PM);
277:  msgRequestCode.version = iVersion;
278:
279: #ifndef __VXWORKS__
280:  pollfd sPoll;
281:  sPoll.fd = iFdReadCode;
282:  sPoll.events = POLLIN | POLLPRI;
283: #else
284:  struct fd_set fds;
285:
286:  timeval timeoutVal;
287:  timeoutVal.tv_sec = 1;
288:  timeoutVal.tv_usec = 0;
289: #endif
290:  while (!bFinished)
291:  {
292:      // Request the code
293:      msgRequestCode.seq_num = usCurSequenceNumber;
294:      msgRequestCode.Send();
295:      MessageSent(&msgRequestCode);
296:      debug_f(2, "Requesting code sequence number %hu \n", usCurSequenceNumber);
297:
298: #ifndef __VXWORKS__
299:      if (poll(&sPoll, 1, 250) == 0)
300: #else
301:      memset(fds.fds_bits, 0, sizeof(fds.fds_bits));
302:      FD_SET(iFdReadCode, &fds);
303:
304:      if(select(iFdReadCode + 1, &fds, NULL, NULL, &timeoutVal) <= 0)
305: #endif
306:      {
307:          if (usPollMiss++ == 10)
308:          {
309:              printf("Code transfer failed -- too many retries. \n");
310:              bValid = false;
311:              break;
312:          }
313:      }
314:
315:      if((iLength = read(iFdReadCode, msgBuffer, sizeof(msgBuffer))) < 0)

```

```

316:     {
317:         perror("Error reading code message \n");
318:         bValid = false;
319:         break;
320:     }
321:
322:     if (msgBuffer[0] != SDM_Code)
323:     {
324:         printf("Received message other than SDMCode, error. \n");
325:         bValid = false;
326:         break;
327:     }
328:     //
329:     // Get the message info
330:     msgCode.Unmarshal(msgBuffer);
331:     usSequenceNumber = msgCode.seq_num;
332:     usNumSegments = msgCode.num_segments;
333:
334:     debug_f(2, " Code message received, sequence number %hu \n", usSequenceNumber);
335:     if (usSequenceNumber != usCurSequenceNumber)
336:         continue;
337:     if (usSequenceNumber > usNumSegments)
338:     {
339:         printf("Fatal error, sequence number is greater than number of segments \n");
340:         fclose(pFile);
341:         return false;
342:     }
343:
344:     //
345:     // Write the code portion
346:     fseek(pFile, (msgCode.seq_num-1)*MAX_CODE_SIZE, SEEK_SET);
347:     fwrite (&msgCode.code, 1, msgCode.code_length, pFile);
348:     if (usCurSequenceNumber++ >= usNumSegments)
349:         bFinished = true;
350:
351:     }
352:     // Perform cleanup, whether success or failure,
353:     // 1. Remove this entry from the code receiver list
354:     // 2. Close the file handle
355:     RemoveCodeReceiver(strFilename);
356:     fclose(pFile);

```

```

357:
358:  if (!bValid)
359:  {
360:      unlink(strFilename);
361:  }
362:  else
363:  {
364: #ifndef __VXWORKS__
365:      // Make the file executable
366:      if(chmod (strFilename, S_IRUSR | S_IWUSR | S_IXUSR |
367:          S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH) != 0)
368:  {
369:      perror("Error changing file permissions to executable \n");
370:  }
371: #endif
372:  }
373:  return bValid;
374: }
375:

```

File: sdm/pm/pm_ids.cpp

```
1: #include <stdio.h>
2: #include <string.h>
3: #include <stdlib.h>
4: #include <sys/wait.h>
5: #include <sys/stat.h>
6: #include <sys/poll.h>
7: #include <errno.h>
8: #include <unistd.h>
9: #include <ctype.h>
10: #include <getopt.h>
11: #include <sys/socket.h>
12: #include <sys/types.h>
13: #include <sys/time.h>
14: #include <netinet/in.h>
15: #include <arpa/inet.h>
16: #include <vector>
17: #include "../common/MessageManipulator/MessageManipulator.h"
18: #include "../common/message/SDMCancel.h"
19: #include "../common/message/SDMReqReg.h"
20: #include "../common/message/SDMRegInfo.h"
21: #include "../common/message/SDMConsume.h"
22: #include "../common/message/SDMData.h"
23: #include "../common/message/SDMCode.h"
24: #include "../common/message/SDMReqCode.h"
25: #include "../common/semaphore/semaphore.h"
26: #include "../common/Debug.h"
27: #include "PMProcess.h"
28: #include "pm.h"
29: #include "pm_main.h"
30:
31: using namespace std;
32:
33: extern SDMMessage_ID tm_mode_msg_id;
34: extern unsigned long Address_PM;
35: extern SDMComponent_ID tm_mode_source;
36: extern int debug;
37: extern bool bgTMDetected;
38: extern unsigned char tm_mode;
39: extern vector<code_rcv> codeReceivers;
```

```

40: extern pthread_mutex_t codeReceiversMutex;
41: extern Sem g_semLaunchTask;
42:
43: // IDS DATA AND DEFINITIONS
44: #define TM_Mode_ID 1 //ID number used in RegInfo and ReqReg messages for TM's status
message
45: #define IDS_OPEN_HANDLE_ID 2
46: #define IDS_CLOSE_HANDLE_ID 3
47: #define IDS_READ_PORTION_ID 4
48:
49: // IDS Data
50: SDMMMessage_ID IDSIDOpenHandle, IDSIDOpenHandleReply, IDSIDCloseHandle,
IDSIDCloseHandleError, IDSIDReadPortion, IDSIDReadPortionReply;
51: MessageManipulator MMIDSOpenHandle, MMIDSCloseHandle, MMIDSReadPortion;
52: SDMComponent_ID IDSCompID;
53: const unsigned char IDS_STATUS_OK = 1; //Status code for OK
54: const unsigned long MAX_PORTION_LENGTH = 24248;
55:
56: pthread_mutex_t IDSFoundMutex = PTHREAD_MUTEX_INITIALIZER;
57: bool g_bIDSFound = false;
58:
59: // New IDS functions
60: bool IDSOpenHandle(const char* strPath, const char *filename, int fd_readcode, unsigned short
*HandleOut);
61: bool IDSReadPortion(unsigned short Handle, unsigned long ByteOffset, unsigned long Length, char
*Buffer, unsigned long *LengthOut, int fd_readcode);
62: bool IDSCloseHandle(unsigned short Handle);
63: bool PipeReadTimeout(int FileDes, char *Buffer, int BufLen, int *LengthReturn);
64: void CancelReqRegSubs(void);
65: PROCESS_HANDLE LaunchTask (char *strTaskFilename, long lSdmPid);
66: // END IDS DATA AND DEFINITIONS
67:
68:
69: enum TaskLocationEnum { TASK_LOCATION_PRIMARY = 1,
TASK_LOCATION_TEMPORARY = 2,
70: TASK_LOCATION_BACKUP = 3 };
71: #ifdef __uLinux__
72: # define STR_TASK_LOCATION_PRIMARY "/primary/"
73: # define STR_TASK_LOCATION_TEMPORARY "/test/"
74: # define STR_TASK_LOCATION_BACKUP "/backup/"
75: # define STR_TASK_LOCATION_DEFAULT "/"
76: #else

```



```

77: # define STR_TASK_LOCATION_PRIMARY "/primary/"
78: # define STR_TASK_LOCATION_TEMPORARY "/test/"
79: # define STR_TASK_LOCATION_BACKUP "/backup/"
80: # define STR_TASK_LOCATION_DEFAULT "/"
81: #endif // #ifdef __uClinux__
82:
83:
84: void HandleCodeMessage(const char* buf)
85: {
86:     debug_f(0, "SDMCode message not handled \n");
87: }
88:
89: /*
90: RegInfoHandler handles the reception of SDMRegInfo messages. The ProcessManager expects to
receive an SDMRegInfo
91: message for the TaskManager's mode message which will be used to reset the PM node upon failure
of the DataManager.
92: INPUTS:
93:     buf - The buffer containing the SDMRegInfo message.
94: RETURNS:
95:     None.
96: */
97: void RegInfoHandler (const char *buf)
98: {
99:     SDMRegInfo msgRegInfo;
100:     SDMConsume msgConsume;
101:     MessageManipulator MM;
102:     static bool IDSReadFound = false, IDSOpenHandleFound = false, IDSCloseHandleFound =
false;
103:
104:     long result = msgRegInfo.Unmarshal(buf);
105:     MessageReceived(&msgRegInfo);
106:
107:     if      (result      ==      SDM_INVALID_MESSAGE      ||      result      ==
SDM_NO_FURTHER_DATA_PROVIDER)
108:         return;
109:
110:     MM.setMsgDef(msgRegInfo);
111:
112:     switch(msgRegInfo.id)
113:     {
114:         case TM_Mode_ID:

```

```

115:     {
116:         debug_f(2,"TM status message found, subscribing. \n");
117:         msgConsume.source = msgRegInfo.source;
118:         msgConsume.msg_id = msgRegInfo.msg_id;
119:         tm_mode_msg_id = msgRegInfo.msg_id.getInterfaceMessagePair();
120:         tm_mode_source = msgRegInfo.source;
121:         msgConsume.destination.setPort(PORT_PM);
122:         msgConsume.Send();
123:         MessageSent(&msgConsume);
124:         /*If an SDMRegInfo is received, as a result of the TM restarting after a fault, reregister
with the TM */
125:         if (bgTMDetected) /*If this is the first series of RegInfo's received, don't reset or re-
register (bgTMDetected is false) */
126:             {
127:                 //DoReset();
128:                 SendRunningTasks();
129:                 RegisterPM();
130:             }
131:         else
132:             bgTMDetected = true; /*The next RegInfo's received will be because of a fault */
133:     }
134:     break;
135:     case IDS_OPEN_HANDLE_ID:
136:     {
137:         if (msgRegInfo.type == SDM_REGINFO_REGISTRATION)
138:         {
139:             IDSIDOpenHandle = MM.getMsgID(COMMANDMSG);
140:             IDSIDOpenHandleReply = MM.getMsgID(DATAMSG);
141:             MMIDSOpenHandle.setMsgDef(msgRegInfo);
142:
143:             debug_f(3,"Found IDS Open Handle \n");
144:
145:             //New IDS found?
146:             if (!(IDSCompID == msgRegInfo.source))
147:             {
148:                 IDSCompID = msgRegInfo.source;
149:             }
150:             IDSOpenHandleFound = true;
151:         }
152:         else if (msgRegInfo.type == SDM_REGINFO_CANCELLATION)
153:         {

```

```

154:         IDSOpenHandleFound = false;
155:         debug_f(3,"IDS open handle message cancelled. \n");
156:     }
157: }
158: break;
159: case IDS_CLOSE_HANDLE_ID:
160: {
161:     if (msgRegInfo.type == SDM_REGINFO_REGISTRATION)
162:     {
163:         IDSIDCloseHandle = MM.getMsgID(COMMANDMSG);
164:         IDSIDCloseHandleError = MM.getMsgID(DATAMSG);
165:         MMIDSCloseHandle.setMsgDef(msgRegInfo);
166:
167:         debug_f(3,"Found IDS Close Handle \n");
168:
169:         //New IDS found?
170:         if (!(IDSCompID == msgRegInfo.source))
171:         {
172:             IDSCompID = msgRegInfo.source;
173:         }
174:         IDSCloseHandleFound = true;
175:     }
176:     else if (msgRegInfo.type == SDM_REGINFO_CANCELLATION)
177:     {
178:         IDSCloseHandleFound = false;
179:         debug_f(3,"IDS close handle message cancelled. \n");
180:     }
181: }
182: break;
183: case IDS_READ_PORTION_ID:
184: {
185:     if (msgRegInfo.type == SDM_REGINFO_REGISTRATION)
186:     {
187:         IDSIDReadPortion = MM.getMsgID(COMMANDMSG);
188:         IDSIDReadPortionReply = MM.getMsgID(DATAMSG);
189:         MMIDSReadPortion.setMsgDef(msgRegInfo);
190:
191:         debug_f(3, "Found IDS Read Portion \n");
192:
193:         //New IDS found?
194:         if (!(IDSCompID == msgRegInfo.source))

```

```

195:         {
196:             IDSCompID = msgRegInfo.source;
197:         }
198:         IDSReadFound = true;
199:     }
200:     else if (msgRegInfo.type == SDM_REGINFO_CANCELLATION)
201:     {
202:         IDSReadFound = false;
203:         debug_f(3,"IDS read portion message cancelled. \n");
204:     }
205: }
206: break;
207: }
208: // If all three IDS services have been found, regard the IDS as available
209: if (IDSReadFound && IDSOpenHandleFound && IDSCloseHandleFound)
210: {
211:     pthread_mutex_lock(&IDSFoundMutex);
212:     g_bIDSFound = true;
213:     pthread_mutex_unlock(&IDSFoundMutex);
214: }
215: else
216: {
217:     pthread_mutex_lock(&IDSFoundMutex);
218:     g_bIDSFound = false;
219:     pthread_mutex_unlock(&IDSFoundMutex);
220: }
221: }
222: /*
223:  * Cancel the PM's registration subscriptions with the Data Manager.
224:  */
225: void CancelReqRegSubs(void)
226: {
227:     SDMCancel msgCancel;
228:
229:     // Cancel the subscription to the TaskManager's mode message
230:     msgCancel.msg_id = tm_mode_msg_id;
231:     msgCancel.source = tm_mode_source;
232:     msgCancel.destination.setPort(PORT_PM);
233:     msgCancel.destination.setAddress(Address_PM);
234:     msgCancel.Send();
235:     MessageSent(&msgCancel);

```

```

236:
237: // Cancel the subscription to the IDS Open Handle
238: msgCancel.msg_id = IDSIDOpenHandle;
239: msgCancel.source = IDSCompID;
240: msgCancel.Send();
241: MessageSent(&msgCancel);
242:
243: // Cancel the subscription to the IDS Close Handle
244: msgCancel.msg_id = IDSIDCloseHandle;
245: msgCancel.Send();
246: MessageSent(&msgCancel);
247:
248: // Cancel the subscription to the IDS Open Handle
249: msgCancel.msg_id = IDSIDReadPortion;
250: msgCancel.Send();
251: MessageSent(&msgCancel);
252: }
253: /*
254:  DataHandler handles the reception of SDMData messages. The ProcessManager expects to
received an SDMData message
255:  containing the TaskManager's current mode. The mode is used to reset the PM node upon failure
of the DataManager.
256:  INPUTS:
257:      buf - The buffer containing the SDMData message.
258:  RETURNS:
259:      None.
260: */
261: void DataHandler (const char *buf)
262: {
263:     SDMData msgData;
264:     SDMComponent_ID newDmId;
265:     unsigned char ucOldMode;
266:     long lResult;
267:     if ((lResult=msgData.Unmarshal(buf)) < 0)
268:     {
269:         printf("Invalid SDMData message. \n");
270:         return;
271:     }
272:     MessageReceived(&msgData);
273:     debug_f(1,"SDMData          message          received          (0x%x).          \n",
msgData.msg_id.getInterfaceMessagePair());
274:

```

```

275: // If this SDMData message is the TaskManager's mode notification
276: if (msgData.source == TaskManager && msgData.msg_id == tm_mode_msg_id)
277: {
278:     // Hang on to the current mode
279:     ucOldMode = tm_mode;
280:     tm_mode = msgData.msg[0];
281:
282:     debug_f(1,"TM Mode change %d. \n",tm_mode);
283:     if (tm_mode == MODE_HARD_RESET)
284:     {
285:         debug_f(1, " Performing a hard reset... \n");
286:         // The DataManager has failed, get the new DM's address from the message
287:         newDmId.Unmarshal(&msgData.msg[1], 0);
288:         DataManager = newDmId;
289:
290:         debug_f(1,"New DM address is 0x%lx:%hd %ld \n",newDmId.getAddress(),
newDmId.getPort(), newDmId.getSensorID());
291:         // Perform a reset
292:         DoReset();
293:         // For a hard reset, reregister the PMs xTEDS
294:         RegisterXteds();
295:     }
296:     else if (tm_mode == MODE_SOFT_RESET)
297:     {
298:         debug_f(1, " Performing a soft reset... \n");
299:         // Perform the reset
300:         DoReset();
301:     }
302:     // Re-set the old mode
303:     tm_mode = ucOldMode;
304: }
305: //
306: // IDS open handle reply
307: else if (msgData.source == IDSCompID && msgData.msg_id == IDSIDOpenHandleReply)
308: {
309:     debug_f(3, "IDSOpenHandle data reply message received. \n");
310:     // Find the file receiver process which corresponds with this filename
311:     unsigned int receiver_index = 0;
312:     int Length;
313:     char *Filename = (char*)MMIDSOpenHandle.getArray("FileName", msgData, DATAMSG,
Length);

```

```

314:     unsigned short IDSHandle = MMIDSOpenHandle.getUINT16Value("FileHandle", msgData,
DATAMSG);
315:
316:     if (Filename == NULL)
317:     {
318:         debug_f(2,"Handle reply message received which does not match any code-consuming
process (filename NULL). \n");
319:         return;
320:     }
321:     //
322:     // Find the "code consuming" processing and pipe this code message
323:     //
324:     pthread_mutex_lock(&codeReceiversMutex);
325:     const unsigned int CODE_RECEIVERS_SIZE = codeReceivers.size();
326:     for (receiver_index = 0; receiver_index < CODE_RECEIVERS_SIZE; receiver_index++)
327:     {
328:         if (!strcmp(codeReceivers[receiver_index].filename, Filename))
329:             break;
330:     }
331:     //
332:     // If no file receiver matched this data message
333:     if (receiver_index == CODE_RECEIVERS_SIZE)
334:     {
335:         debug_f(2,"Handle reply message received which does not match any code-consuming
process. \n");
336:         pthread_mutex_unlock(&codeReceiversMutex);
337:         return;
338:     }
339:     codeReceivers[receiver_index].ids_handle = IDSHandle;
340:     //
341:     // Write the code message to the pipe
342:     if (write(codeReceivers[receiver_index].fd_code, buf, lResult) <= 0)
343:         perror("Error writing code message to code pipe in HandleCodeMessage(). \n");
344:     pthread_mutex_unlock(&codeReceiversMutex);
345: }
346: // IDS read portion reply
347: else if (msgData.source == IDSCompID && msgData.msg_id == IDSIDReadPortionReply)
348: {
349:     debug_f(3, "IDSReadPortion data reply message received. \n");
350:     // Find the file receiver process which corresponds to this handle
351:     unsigned int receiver_index = 0;

```

```

352:     unsigned short MsgHandle = MMIDSReadPortion.getUINT32Value("FileHandle", msgData,
DATAMSG);
353:
354:     pthread_mutex_lock(&codeReceiversMutex);
355:     const unsigned int CODE_RECEIVERS_SIZE = codeReceivers.size();
356:     for (receiver_index = 0; receiver_index < CODE_RECEIVERS_SIZE; receiver_index++)
357:     {
358:         if (codeReceivers[receiver_index].ids_handle == MsgHandle)
359:         {
360:             debug_f(2, "Code message received for %s \n",
codeReceivers[receiver_index].filename);
361:             break;
362:         }
363:     }
364:     // If no file receiver matched this data message
365:     if (receiver_index == CODE_RECEIVERS_SIZE)
366:     {
367:         debug_f(2,"ReadPortion reply message received which does not match any code-
consuming process. \n");
368:         pthread_mutex_unlock(&codeReceiversMutex);
369:         return;
370:     }
371:     // Write the code message to the pipe
372:     else if (write(codeReceivers[receiver_index].fd_code, buf, lResult) <= 0)
373:         perror("Error writing code message to code pipe in HandleCodeMessage(). \n");
374:     pthread_mutex_unlock(&codeReceiversMutex);
375: }
376: }
377:
378: /*
379: The GetCode routine requests executable code from the TM
380: INPUTS:
381:     filename - name of the code desired
382:     fd_readcode - file descriptor for the pipe to read the code messenger
383: RETURN VALUE:
384:     true - code successfully retrieved
385:     false - code not retrieved
386: */
387: bool GetCode (const char* filename, long pid, int iVersion, int fd_readcode)
388: {
389:     bool IsValid=false; //true if all CRC checks on the code were successful
390:     FILE *fp;          //a file pointer to write to

```



```

391:  char FileBuf[BUFSIZE];
392:  bool FileReceived = false;
393:  unsigned short IDSHandle;
394:
395:  debug_f(2, "  requesting code (%s)... \n", filename);
396:  //
397:  // If the IDS has not yet registered, wait until it does
398:  pthread_mutex_lock(&IDSFoundMutex);
399:  bool bIDSFound = g_bIDSFound;
400:  pthread_mutex_unlock(&IDSFoundMutex);
401:  while (!bIDSFound)
402:  {
403:      sleep(1);
404:      pthread_mutex_lock(&IDSFoundMutex);
405:      bIDSFound = g_bIDSFound;
406:      pthread_mutex_unlock(&IDSFoundMutex);
407:  }
408:  //
409:  // Determine which path to transfer from
410:  char strPath[128];
411:  switch (iVersion)
412:  {
413:      case TASK_LOCATION_PRIMARY:
414:          strncpy(strPath, STR_TASK_LOCATION_PRIMARY, sizeof(strPath));
415:          debug_f(3, "  Using primary location %s to retrieve task. \n", strPath);
416:          break;
417:      case TASK_LOCATION_TEMPORARY:
418:          strncpy(strPath, STR_TASK_LOCATION_TEMPORARY, sizeof(strPath));
419:          debug_f(3, "  Using temporary location %s to retrieve task. \n", strPath);
420:          break;
421:      case TASK_LOCATION_BACKUP:
422:          strncpy(strPath, STR_TASK_LOCATION_BACKUP, sizeof(strPath));
423:          debug_f(3, "  Using backup location %s to retrieve task. \n", strPath);
424:          break;
425:      default:
426:          strncpy(strPath, STR_TASK_LOCATION_DEFAULT, sizeof(strPath));
427:          debug_f(3, "  Using default location %s to retrieve task. \n", strPath);
428:      }
429:      strPath[sizeof(strPath) - 1] = '\0';
430:
431:      //Obtain handle

```

```

432:     if (!IDSOpenHandle(strPath, filename, fd_readcode, &IDSHandle))
433:         return false;
434:
435:     //Open file
436:     g_semLaunchTask.Wait();
437:     fp = fopen(filename, "wb");
438:     if (SetCloseFileOnExec(fileno(fp)) == -1)
439:         printf("Error setting new file to close on fork. \n");
440:     g_semLaunchTask.Signal();
441:
442:     unsigned long CurBytesReceived = 0, TotalBytesReceived = 0;
443:     unsigned long Offset = 0;
444:     int FailCount = 0;
445:     const int READ_ATTEMPTS = 10;
446:     while (!FileReceived)
447:     {
448:         memset(FileBuf, 0, sizeof(FileBuf));
449:         // Request a read portion from the IDS
450:         if (false == IDSReadPortion(IDSHandle, Offset, MAX_PORTION_LENGTH,
451:             FileBuf, &CurBytesReceived, fd_readcode))
452:         {
453:             debug_f(3, " (%s) IDSReadPortion failed. Trying %d more times \n",
454:                 filename, READ_ATTEMPTS - FailCount);
455:             if (FailCount++ >= READ_ATTEMPTS)
456:                 break;
457:         }
458:         else
459:         {
460:             FailCount = 0;
461:             if (CurBytesReceived == 0)
462:             {
463:                 // If file reading is done
464:                 FileReceived = true;
465:                 IsValid = true;
466:                 break;
467:             }
468:
469:             debug_f(2, " (%s) IDS Read success. Writing %lu bytes to file \
470:                 at offset %lu. \n", filename, CurBytesReceived, Offset);
471:             debug_f(3, " (%s) CurBytesReceived is %lu \n", filename,
472:                 CurBytesReceived);

```

```

473:
474:     if (fwrite(FileBuf, 1, CurBytesReceived, fp) == 0)
475:     {
476:         printf ("Error with fwrite. \n");
477:     }
478:     TotalBytesReceived += CurBytesReceived;
479:     Offset += CurBytesReceived;
480:
481:     debug_f(3, " (%s) Total bytes received is %lu \n",
482:         filename, TotalBytesReceived);
483: }
484: }
485: // Code transferring has finished, whether success or error, perform cleanup
486: // 1. Remove this code receiver entry
487: RemoveCodeReceiver (filename);
488:
489: // 2. Close the IDS handle
490: IDSCloseHandle(IDSHandle);
491:
492: // 3. Close the file
493: fclose (fp);
494:
495: if (IsValid)
496: {
497:     debug_f(3, " (%s) Code retrieval success. \n",filename);
498:     // Make the file executable
499:     chmod (filename, S_IRUSR | S_IWUSR | S_IXUSR |
500:         S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH);
501:     return true;
502: }
503: }
504: else //delete bad code
505: {
506:     debug_f(3, " (%s) Code retrieval failed. \n",filename);
507:     unlink(filename);
508:     return false;
509: }
510: return false; //this point should not be reached
511: }
512:
513: /*

```

```

514: * Read a portion of a file from the IDS and store it in Buffer (must be at least BUFSIZE)
515: */
516: bool IDSReadPortion(unsigned short Handle, unsigned long ByteOffset, unsigned long Length, char
*Buffer, unsigned long *LengthOut, int fd_readcode)
517: {
518:     unsigned short IDSHandleReturned = 0;
519:     unsigned long IDSLengthReturned = 0;
520:     unsigned long IDSOffsetReturned = 0;
521:     unsigned char IDSStatusReturned = 0;
522:     void *IDSFileBufferReturned = NULL;
523:     int IDSFileBufferLength = 0;
524:     char MsgBuf[BUFSIZE];
525:
526:     if (Buffer == NULL || LengthOut == NULL)
527:         return false;
528:
529:     SDMSERVICE Request;
530:     Request.command_id = IDSIDReadPortion;
531:     Request.destination.setPort(PORT_PM);
532:     Request.source = IDSCompID;
533:     if ( !MMIDSReadPortion.setValue("FileHandle", Request, Handle)
534:         || !MMIDSReadPortion.setValue("ByteOffset", Request, ByteOffset)
535:         || !MMIDSReadPortion.setValue("Length", Request, Length) )
536:     {
537:         printf("IDSReadPortion::Error with MessageManipulator. \n");
538:         return false;
539:     }
540:     Request.length = MMIDSReadPortion.getLength(COMMANDMSG);
541:     Request.Send();
542:
543:     int MsgLength;
544:     if (!PipeReadTimeout(fd_readcode, MsgBuf, sizeof(MsgBuf), &MsgLength))
545:         return false;
546:
547:     SDMDATA Reply;
548:     if (Reply.Unmarshal(MsgBuf) < 0)
549:         return false;
550:     //
551:     // Some error checking to make sure we received a valid message
552:     IDSHandleReturned      =      MMIDSReadPortion.getUINT16Value("FileHandle",      Reply,
DATAMSG);

```

```

553:  if (IDSHandleReturned != Handle)
554:  {
555:      debug_f(1, "%s -- IDS handle returned is invalid. \n", __FUNCTION__);
556:      return false;
557:  }
558:
559:  IDSStatusReturned = MMIDSReadPortion.getUINT08Value("StatusCode", Reply, DATAMSG);
560:  if (IDSStatusReturned != IDS_STATUS_OK)
561:  {
562:      debug_f(1, "%s -- IDS status returned is invalid. \n", __FUNCTION__);
563:      return false;
564:  }
565:
566:  IDSLengthReturned = MMIDSReadPortion.getUINT32Value("Length", Reply, DATAMSG);
567:  if (IDSLengthReturned > BUFSIZE)
568:  {
569:      debug_f(1, "%s -- IDS length returned is invalid. \n", __FUNCTION__);
570:      return false;
571:  }
572:
573:  IDSOffsetReturned = MMIDSReadPortion.getUINT32Value("ByteOffset", Reply, DATAMSG);
574:  if (IDSOffsetReturned != ByteOffset)
575:  {
576:      debug_f(1, "%s -- IDS offset returned is invalid. \n", __FUNCTION__);
577:      return false;
578:  }
579:
580:  IDSFileBufferReturned = MMIDSReadPortion.getArray("FileBuffer", Reply, DATAMSG,
IDSFileBufferLength);
581:  if (IDSFileBufferReturned == NULL)
582:  {
583:      debug_f(1, "%s -- IDS file buffer returned is invalid. \n", __FUNCTION__);
584:      return false;
585:  }
586:
587:  // Return the buffer and the length
588:  memcpy(Buffer, IDSFileBufferReturned, IDSLengthReturned);
589:  *LengthOut = IDSLengthReturned;
590:  return true;
591: }
592:

```

```

593: /*
594:  * Request a handle for the specified file from the IDS.
595:  */
596: bool IDSOpenHandle(const char* strPath, const char* filename, int fd_readcode, unsigned short
*HandleOut)
597: {
598:     unsigned short IDSHandle = 0;
599:     const unsigned char FileMode = 1;        //IDS Filemode ReadOnly
600:     unsigned char IDSStatus = '\0';
601:     int length;
602:     char buf[BUFSIZE];
603:     const int NUM_RETRIES = 10;
604:
605:     debug_f(2, "IDSOpenHandle for filename %s \n", filename);
606:
607:     if (HandleOut == NULL)
608:         return false;
609:     //
610:     // Request a file handle for this file
611:     SDMSservice Request;
612:     Request.command_id = IDSIDOpenHandle;
613:     Request.destination.setPort(PORT_PM);
614:     Request.source = IDSCompID;
615:
616:     if ( !MMIDSOpenHandle.setArray("PathName", Request, strPath, strlen(strPath)+1)
617:         || !MMIDSOpenHandle.setArray("FileName", Request, filename, strlen(filename)+1)
618:         || !MMIDSOpenHandle.setValue("FileFlag", Request, FileMode) )
619:     {
620:         printf("IDSFileRequest::Error with MessageManipulator. \n");
621:         return false;
622:     }
623:     Request.length = MMIDSOpenHandle.getLength(COMMANDMSG);
624:
625:     int iSendCount = 0;
626:     do
627:     {
628:         Request.Send();
629:         // Read from the pipe
630:         if (false == PipeReadTimeout(fd_readcode, buf, sizeof(buf), &length))
631:         {
632:             if (length == 0)

```

```

633:         {
634:             // Read timeout
635:             iSendCount++;
636:         }
637:     else
638:     {
639:         // Error
640:         return false;
641:     }
642: }
643: else
644: {
645:     // Read success
646:     break;
647: }
648: } while (iSendCount < NUM_RETRIES);
649:
650: if (length <= 0)
651: {
652:     // Read error or no response
653:     return false;
654: }
655:
656: SDMDData Reply;
657: Reply.Unmarshal(buf);
658: IDSSStatus = MMIDSOpenHandle.getUINT08Value("StatusCode", Reply, DATAMSG);
659: debug_f(2, "IDSOpenHandle reply, status is 0x%hhx \n", IDSSStatus);
660:
661: if (IDSSStatus != IDS_STATUS_OK)
662: {
663:     //File probably doesn't exist
664:     return false;
665: }
666: //
667: // Get and return the handle
668: IDSHandle = MMIDSOpenHandle.getUINT16Value("FileHandle", Reply, DATAMSG);
669: if (IDSHandle == 0)
670:     return false;
671:
672: *HandleOut = IDSHandle;
673: return true;

```

```

674: }
675:
676: bool IDSCloseHandle(unsigned short Handle)
677: {
678:     SDMCommand Command;
679:     Command.command_id = MMIDSCloseHandle.getMsgID(COMMANDMSG);
680:     if ( !MMIDSCloseHandle.setValue("FileHandle", Command, Handle) )
681:     {
682:         printf("IDSCloseHandle::Error with MessageManipulator. \n");
683:         return false;
684:     }
685:     Command.length = MMIDSCloseHandle.getLength(COMMANDMSG);
686:     Command.destination.setPort(PORT_PM);
687:     Command.source = IDSCompID;
688:     Command.Send();
689:     return true;
690: }
691:
692: void ReqRegSubs(void)
693: {
694:     SDMReqReg Request;
695:
696:     //Request the IDS open handle service
697:     Request.destination.setPort(PORT_PM);
698:     Request.id = IDS_OPEN_HANDLE_ID;
699:     strcpy(Request.interface, "FileAccessInterface");
700:     strcpy(Request.item_name, "OpenFileHandle");
701:     Request.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
702:     Request.Send();
703:     MessageSent(&Request);
704:
705:     //Request the IDS close handle service
706:     Request.id = IDS_CLOSE_HANDLE_ID;
707:     strcpy(Request.item_name, "CloseFileHandle");
708:     Request.Send();
709:     MessageSent(&Request);
710:
711:     //Request the IDS read portion service
712:     Request.id = IDS_READ_PORTION_ID;
713:     strcpy(Request.item_name, "ReadPortion");
714:     Request.Send();

```



```

715:   MessageSent(&Request);
716:
717:   //SDMReqReg for TM's status message
718:   debug_f(2,"Looking for TM status message. \n");
719:   Request.reply = SDM_REQREG_CURRENT_AND_FUTURE;
720:   Request.id = TM_Mode_ID;
721:   strcpy(Request.device, "TaskManager");
722:   strcpy(Request.interface, "TM_Interface");
723:   strcpy(Request.item_name, "Status");
724:   Request.destination.setAddress(Address_PM);
725:   Request.destination.setPort(PORT_PM);
726:   Request.Send();
727:   MessageSent(&Request);
728: }
729: /*
730:  * Read a SDM message from a pipe, with a five second timeout imposed.
731:  * Params:
732:  *     FileDes - The pipe file descriptor from which to read
733:  *     Buffer - The buffer into which to store the message
734:  *     BufLen - The size in bytes of Buffer, must be at least BUFSIZE
735:  *     LengthReturn - [Output] The number of bytes stored into Buffer, 0 if timeout, -1 if error
736:  * Returns:
737:  *     bool - True if the read succeeds, false otherwise or upon timeout
738:  */
739: bool PipeReadTimeout(int FileDes, char *Buffer, int BufLen, int *LengthReturn)
740: {
741:   int BytesRead = 0, MsgLength = BUFSIZE, ReadResult = 0;
742:   struct pollfd ufd;
743:   ufd.fd = FileDes;
744:   ufd.events = POLLIN | POLLPRI;
745:   bool ReadDone = false;
746:
747:   if (BufLen < BUFSIZE)
748:   {
749:     *LengthReturn = -1;
750:     return false;
751:   }
752:   memset(Buffer, 0, BufLen);
753:   // Wait for the reply
754:   bool HeaderReceived = false;
755:   while (!ReadDone)

```

```

756:  {
757: #ifndef WIN32
758:     // Wait 5 seconds for a message
759:     // Win32 doesn't support poll for pipe file descriptors, what should be done here is
760:     // TBD. No timeout may be OK for our purposes in Win32.
761:     if (poll (&ufd, 1U, 5000) > 0)
762: #endif
763:     {
764:         if (!HeaderReceived)
765:         {
766:             BytesRead += ReadResult = read(FileDes, Buffer + BytesRead, HEADER_SIZE -
BytesRead);
767:             if (ReadResult < 0)
768:             {
769:                 *LengthReturn = -1;
770:                 return false;
771:             }
772:             if (BytesRead < HEADER_SIZE)
773:                 continue;
774:             // Get the message length of the SDM message
775:             MsgLength = GET_USHORT(Buffer+9) + HEADER_SIZE;
776:             if (MsgLength > BUFSIZE || MsgLength < 0)
777:             {
778:                 *LengthReturn = -1;
779:                 return false;
780:             }
781:             HeaderReceived = true;
782:         }
783:         BytesRead += ReadResult = read(FileDes, Buffer + BytesRead, MsgLength -
BytesRead);
784:         if (ReadResult < 0)
785:         {
786:             *LengthReturn = -1;
787:             return false;
788:         }
789:         if (MsgLength - BytesRead <= 0)
790:             ReadDone = true;
791:     }
792: #ifndef WIN32
793:     else
794:     {

```

```
795:         // Timeout
796:         *LengthReturn = 0;
797:         return false;
798:     }
799: #endif
800: }
801: *LengthReturn = BytesRead;
802: return true;
803: }
804:
```

File: sdm/pm/HeartbeatTestApp.cpp

```
1: #include "../common/message/SDMHeartbeat.h"
2: #include <stdio.h>
3: #include <unistd.h>
4: #include <stdlib.h>
5: #include <sys/types.h>
6:
7: int main(int argc, char** argv)
8: {
9:     SDMInit(argc, argv);
10:
11:     printf("Heartbeat test app running (%d). \n", getpid());
12:
13:     SendHeartbeat();
14:
15:     while (1)
16:     {
17:         sleep(1);
18:     }
19:     return 0;
20: }
```

File: sdm/pm/pmxTEDS.h

```
1: #ifndef _SDM_PM_XTEDS_H_
2: #define _SDM_PM_XTEDS_H_
3:
4: const char * xTEDS = "<?xml version= \"1.0\" encoding= \"UTF-8\"?> \n \
5: <xTEDS version= \"2.0\" xmlns= \"http://www.interfacecontrol.com/SPA/xTEDS\" xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance\" xsi:schemaLocation=
\"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd\" name=
\"Process_Manager_xTEDS\"> \n \
6:   <Application kind= \"Software\" name= \"ProcessManager\"/> \n \
7:   <Interface name= \"Msg_Count\" id= \"2\"> \n \
8:     <Variable name= \"Total_Messages_Recd\" kind= \"Total\" format= \"UINT32\"/> \n \
9:     <Variable name= \"Messages_Last_Second_Recd\" kind= \"Total\" format= \"UINT32\"/> \n \
10:    <Variable name= \"Total_Messages_Sent\" kind= \"Total\" format= \"UINT32\"/> \n \
11:    <Variable name= \"Messages_Last_Second_Sent\" kind= \"Total\" format= \"UINT32\"/> \n \
12:   \n \
13:   <Notification> \n \
14:     <DataMsg name= \"Message_Count\" id= \"13\" msgArrival= \"PERIODIC\"> \n \
15:       <VariableRef name= \"Total_Messages_Recd\"/> \n \
16:       <VariableRef name= \"Messages_Last_Second_Recd\"/> \n \
17:       <VariableRef name= \"Total_Messages_Sent\"/> \n \
18:       <VariableRef name= \"Messages_Last_Second_Sent\"/> \n \
19:     </DataMsg> \n \
20:   </Notification> \n \
21: </Interface> \n \
22: <Interface name= \"Message_Log\" id= \"3\"> \n \
23:   <Variable format= \"UINT08\" kind= \"TBD\" name= \"Msg_Type\"/> \n \
24:   <Variable format= \"UINT32\" kind= \"IP_long\" name= \"Address\"/> \n \
25:   <Variable format= \"UINT16\" kind= \"Port_of_Device\" name= \"Port\"/> \n \
26:   <Variable format= \"UINT32\" kind= \"ID\" name= \"Sensor_ID\"/> \n \
27:   <Command> \n \
28:     <CommandMsg name= \"Enable_Logging\" id= \"16\"> \n \
29:       <VariableRef name= \"Msg_Type\"/> \n \
30:       <VariableRef name= \"Address\"/> \n \
31:       <VariableRef name= \"Port\"/> \n \
32:       <VariableRef name= \"Sensor_ID\"/> \n \
33:     </CommandMsg> \n \
34:   </Command> \n \
35:   <Command> \n \
36:     <CommandMsg name= \"Disable_Logging\" id= \"17\"> \n \
37:       <VariableRef name= \"Msg_Type\"/> \n \
```

```
38: \t \t \t \t<VariableRef name= \"Address \"/> \n \
39: \t \t \t \t<VariableRef name= \"Port \"/> \n \
40: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
41: \t \t \t</CommandMsg> \n \
42: \t \t</Command> \n \
43: \t</Interface> \n \
44: </xTEDS>";
45:
46: #endif
```

File: sdm/pm/pm_monitor.cpp

```
1: #include <stdio.h>
2: #include <unistd.h>
3: #include <sys/types.h>
4: #include <signal.h>
5: #include <stdlib.h>
6: #include <arpa/inet.h>
7: #include <netinet/in.h>
8: #include <sys/socket.h>
9: #include <sys/wait.h>
10: #include <string.h>
11: #include "../common/message/SDMReady.h"
12: #include "../common/MessageManager/MessageManager.h"
13: #include "../common/TCPcom.h"
14: #include "../common/message/SDMHeartbeat.h"
15: #include "../common/Time/SDMTime.h"
16:
17: #define NUM_HEARTBEAT_TRIES    10 /*Number of times to miss a response before restarting
the PM*/
18:
19: int StartPM(char **);
20: void SigIntHandler(int);
21:
22: int pm_pid = -1;    //Process id number of the Process Manager
23:
24: int main(int argc, char ** argv)
25: {
26:     int result = -1;
27:     int miss_count = 0;
28:     int num_toget = 0;
29:     char buf[BUFSIZE];
30:     SDMHeartbeat heartbeat;
31:     MessageManager mm;
32:     SDMComponent_ID pm;
33:
34:     if (argc == 1)
35:     {
36:         printf("Usage: \n%s pm_process [pm_process_args...] \n", argv[0]);
37:         return -1;
38:     }
```

```

39: // Set up the command line for the child process
40: char **nargv = new char*[argc];
41: for (int i = 1; i < argc; ++i)
42:     nargv[i-1] = argv[i];
43: nargv[argc-1] = NULL;
44:
45: //Set monitor process address
46: heartbeat.source.setSensorID(0);
47: heartbeat.source.setAddress(inet_addr("127.0.0.1"));
48: heartbeat.source.setPort(PORT_PM_MONITOR);
49:
50: //Set PM address
51: pm.setAddress(inet_addr("127.0.0.1"));
52: pm.setPort(PORT_PM);
53: pm.setSensorID(0);
54:
55: signal(SIGINT, SigIntHandler);
56: pm_pid = StartPM(nargv);
57: mm.Async_Init(PORT_PM_MONITOR);
58:
59: //If fork/exec was successful
60: if (pm_pid > 0)
61: {
62:     //Allow the PM to get started up
63:     sleep(HEARTBEAT_INTERVAL);
64:     while (1)
65:     {
66:
67:         //Send heartbeats via UDP and TCP
68:         heartbeat.SendTo(pm);
69:         sleep(HEARTBEAT_INTERVAL);
70:
71:         //If Process Manager quit
72:         if (waitpid(-1,&result,WNOHANG) == pm_pid)
73:         {
74:             printf(" Monitor: PM failed, restarting... \n");
75:             pm_pid = StartPM(nargv);
76:             if (pm_pid > 0)
77:             {
78:                 sleep(HEARTBEAT_INTERVAL);
79:                 continue;

```



```

80:         }
81:     else
82:     {
83:         printf(" Monitor: Could not restart the PM. \n");
84:         return -1;
85:     }
86: }
87: else if (mm.IsReady())
88: {
89:     //Should respond with one message
90:     num_toget = 1;
91:
92:     while (mm.IsReady())
93:     {
94:         num_toget--;
95:         mm.GetMessage(buf);
96:     }
97:     if (num_toget > 0)
98:         miss_count++;
99:     else
100:         miss_count = 0;
101: }
102: else
103: {
104:     //If no messages were received
105:     miss_count++;
106:     if (miss_count == NUM_HEARTBEAT_TRIES)
107:     {
108:         printf(" Monitor: PM unresponsive, restarting... \n");
109:
110:         // Be nice first
111:         kill(pm_pid, SIGINT);
112:         usleep(10000);
113:
114:         // Then be mean
115:         if (kill (pm_pid, SIGKILL) == 0)
116:             wait(&result);
117:         else
118:             waitpid(-1, &result, WNOHANG);
119:
120:         pm_pid = StartPM(nargv);

```

```

121:         if (pm_pid > 0)
122:         {
123:             sleep(HEARTBEAT_INTERVAL);
124:             continue;
125:         }
126:         else
127:         {
128:             printf(" Monitor: Could not restart the PM. \n");
129:             return -1;
130:         }
131:
132:
133:     }
134: }
135: }
136: }
137: else
138: {
139:     printf(" Monitor: Error starting the PM (%d). \n", pm_pid);
140:     return -1;
141: }
142: return 0;
143: }
144:
145: int StartPM(char ** argv)
146: {
147:     int pid;
148:
149:     pid = vfork();
150:     //Child Process
151:     if (pid == 0)
152:     {
153:         //Start the Process Manager
154:         if (execvp(argv[0],argv) < 0)
155:         {
156:             printf(" Monitor: Error exec'ing the PM. \n");
157:             exit(-1);
158:         }
159:         //Here to make the compiler happy, never runs however
160:         return 0;
161:     }

```

```
162: //Parent Process
163: else
164:     return pid;
165: }
166:
167: void SigIntHandler(int sig_num)
168: {
169:     int result = 0;
170:     if (kill (SIGINT, pm_pid) == 0)
171:         wait (&result);
172:     else
173:         waitpid(-1,&result,WNOHANG);
174:
175:     exit(EXIT_SUCCESS);
176: }
177:
```

File: sdm/pm/PMProcess.h

```
1: #ifndef _SDM_PM_PROCESS_H_
2: #define _SDM_PM_PROCESS_H_
3:
4: #include "../common/message_defs.h"
5: #include "../common/version.h"
6:
7: #ifdef WIN32
8: # include <windows.h>
9: # define PROCESS_HANDLE HANDLE
10: #else
11: # define PROCESS_HANDLE int
12: #endif
13:
14: #define TASK_HEARTBEAT_TIMEOUT      10 // In seconds
15: #define TASK_TIMEOUT_ATTEMPTS      6  // Number of retries
16:
17: class PMProcess
18: {
19: public:
20: PMProcess();
21: PMProcess(const char* Name, unsigned int SdmPid, PROCESS_HANDLE OsPid);
22: ~PMProcess();
23:
24: unsigned int GetSdmPid() const { return m_uiSdmPid; }
25: PROCESS_HANDLE GetOsPid() const { return m_OsPid; }
26: const char* GetName() const { return m_strFilename; }
27: unsigned int GetFailCount() const { return m_uiTotalFailCount; }
28: void SetOsPid(PROCESS_HANDLE OsPid) { m_OsPid = OsPid; }
29: bool Kill() const;
30: bool IsNormalFinish() const { return !m_bFailedState; }
31: bool IsUnresponsive();
32: bool IsFailed() const { return m_bFailedState; }
33: void Failure();
34: void Reset();
35: void HeartbeatReceived();
36:
37: #ifdef WIN32
38: void SetCode(bool NewCode) { m_bCode = NewCode; }
39: bool GetCode() const { return m_bCode; }
```

```

40: #endif
41: private:
42: char m_strFilename[MAX_FILENAME_SIZE];    // Process name
43: unsigned long m_ulLastHeartbeat;          // Seconds of last heartbeat, in EPOCH time
44: unsigned long m_ulHeartbeatMissCount;      // Number of heartbeats missed
45: unsigned int m_uiSdmPid;                   // SDM PID for this process
46: PROCESS_HANDLE m_OsPid;                   // Operating System PID for this process
47: unsigned int m_uiTotalFailCount;          // Number of times this process has failed
48: bool m_bFailedState;                     // Indicates whether this process is in a fail state
49: #ifdef WIN32
50: bool m_bCode;
51: #endif
52: static unsigned int GetCurTimeSeconds();
53: static bool m_bTimerInitialized;
54: };
55:
56: #endif

```

File: sdm/pm/PendingTask.h

```
1: #ifndef __SDM_PENDING_TASK_H_
2: #define __SDM_PENDING_TASK_H_
3:
4: #include "../common/message_defs.h"
5:
6: class PendingTask
7: {
8: public:
9:     PendingTask();
10:    PendingTask(const char* strFilename, unsigned long ulSdmPid);
11:    PendingTask(const PendingTask& right);
12:    PendingTask& operator= (const PendingTask& right);
13:
14:    char* GetName() { return m_strName; }
15:    unsigned long GetPid() const { return m_ulSdmPid; }
16: private:
17:    char m_strName[MAX_FILENAME_SIZE];
18:    unsigned long m_ulSdmPid;
19: };
20:
21: #endif
22:
```

File: sdm/pm/Makefile.uclinux

```
1: ifndef PETALINUX
2: $(error You must source the petalinux/settings.sh script before working with PetaLinux)
3: endif
4:
5: # Point to default PetaLinux root directory
6: ifndef ROOTDIR
7: ROOTDIR=$(PETALINUX)/software/petalinux-dist
8: endif
9:
10: PATH:=$(PATH):$(ROOTDIR)/tools
11:
12: UCLINUX_BUILD_USER = 1
13: -include $(ROOTDIR)/.config
14: -include $(ROOTDIR)/$(CONFIG_LINUXDIR)/.config
15: LIBCDIR = $(CONFIG_LIBCDIR)
16: -include $(ROOTDIR)/config.arch
17: ROMFSDIR=$(ROOTDIR)/romfs
18: ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh
19:
20: PM = pm
21: PM_IDS = pm_ids
22: PM_MONITOR = pm_monitor
23:
24: # Add any other object files to this list below
25: PM_IDS_OBJS = pm_ids.o pm_main_ids.o PMProcess.o PMProcessList.o PendingTask.o
26: PM_OBJS = pm.o pm_main.o PMProcess.o PMProcessList.o PendingTask.o
27: PM_MONITOR_OBJS = pm_monitor.o
28:
29: FLTFLAGS+=-s 262144
30: export FLTFLAGS
31:
32: LDLIBS += -lSDM -lpthread -lstdc++
33: LDFLAGS += -L../common/
34: all: $(PM) $(PM_IDS) $(PM_MONITOR)
35:
36: $(PM): $(PM_OBJS)
37: $(CXX) $(LDFLAGS) -o $@ $(PM_OBJS) $(LDLIBS)
38: cp $@ ../FLIGHT_BINARIES/
39:
```

```

40: $(PM_IDS): $(PM_IDS_OBJS)
41: $(CXX) $(LDFLAGS) -o $@ $(PM_IDS_OBJS) $(LDLIBS)
42: cp $@ ../../FLIGHT_BINARIES/
43:
44: $(PM_MONITOR): $(PM_MONITOR_OBJS)
45: $(CXX) $(LDFLAGS) -o $@ $(PM_MONITOR_OBJS) $(LDLIBS)
46: cp $@ ../../FLIGHT_BINARIES/
47:
48: pm_main_ids.o: pm_main.cpp pm.h
49: $(CXX) -c $(CXXFLAGS) -DBUILD_FOR_IDS -o $@ $<
50:
51: pm_ids.o: pm_ids.cpp pm_ids.h
52: $(CXX) -c $(CXXFLAGS) -DBUILD_FOR_IDS -o $@ $<
53:
54: clean:
55: -rm -f $(PM) $(PM_IDS) $(PM_MONITOR) *.elf *.gdb *.o
56:
57: romfs:
58: $(ROMFSINST) $(PM) /bin/$(PM)
59: $(ROMFSINST) $(PM_IDS) /bin/$(PM_IDS)
60: $(ROMFSINST) $(PM_MONITOR) /bin/$(PM_MONITOR)
61:
62: %.o: %.cpp
63: $(CXX) -c $(CXXFLAGS) -o $@ $<
64:
65:
66: # Targets for the required .config files - if they don't exist, the tree isn't
67: # configured. Tell the user this, how to fix it, and exit.
68: ${ROOTDIR}/config.arch ${ROOTDIR}/.config:
69: @echo "Error: You must configure the PetaLinux tree before compiling your application"
70: @echo ""
71: @echo "Change directory to ../../petalinux-dist and 'make menuconfig' or 'make xconfig'"
72: @echo ""
73: @echo "Once the tree is configured, return to this directory, and re-run make."
74: @echo ""
75: @exit -1
76:

```


File: sdm/pm/Makefile

```
1: # Makefile for pm system
2:
3: include ../Makefile.common
4: include ../$(MAKEFILE_DEFS)
5:
6: .PHONY:    clean distclean
7:
8: all: pm pm_ids pm_monitor
9:
10: # Version of the PM to use the TM
11: pm: pm_main.o pm.o PMProcess.o PMProcessList.o PendingTask.o
12: $(CXX) $(CXXFLAGS) -L../common -static -o $$@ $$^ -lSDM -lpthread
13:
14: # Version of the PM to use the IDS
15: pm_ids: pm_ids.o pm_main_ids.o PMProcess.o PMProcessList.o PendingTask.o
16: $(CXX) $(CXXFLAGS) -L../common -static -o $$@ $$^ -lSDM -lpthread
17:
18: pm_monitor: pm_monitor.o
19: $(CXX) $(CXXFLAGS) -L../common -static -o $$@ $$^ -lSDM -lpthread
20:
21: PMProcess.o: PMProcess.cpp PMProcess.h
22: $(CXX) $(CXXFLAGS) -c $$<
23:
24: PMProcessList.o: PMProcessList.cpp PMProcessList.h
25: $(CXX) $(CXXFLAGS) -c $$<
26:
27: PendingTask.o: PendingTask.cpp PendingTask.h
28: $(CXX) $(CXXFLAGS) -c $$<
29:
30: pm_main_ids.o: pm_main.cpp pm.h
31: $(CXX) $(CXXFLAGS) -c -o $$@ -DBUILD_FOR_IDS $$<
32:
33: pm_main.o: pm_main.cpp pm.h
34: $(CXX) $(CXXFLAGS) -c $$<
35:
36: pm.o: pm.cpp pm.h
37: $(CXX) $(CXXFLAGS) -c $$<
38:
39: pm_ids.o: pm_ids.cpp pm_ids.h
```

```
40: $(CXX) $(CXXFLAGS) -c -DBUILD_FOR_IDS $<
41:
42: pm_monitor.o: pm_monitor.cpp
43: $(CXX) $(CXXFLAGS) -c $<
44:
45: #%.o: %.cpp %.h
46: # $(CXX) $(CXXFLAGS) -c $<
47:
48: clean:
49: rm -f *.o *.log SDMMessages*
50: rm -f consumer converter producer
51:
52: distclean: clean
53: rm -f pm pm_ids *~ pm_monitor
```

File: sdm/pm/pm_main.h

```
1: #ifndef _SDM_PM_H_
2: #define _SDM_PM_H_
3:
4: #include <stdlib.h>
5: #include <stdio.h>
6: #include <string.h>
7: #include <sys/wait.h>
8: #include <sys/stat.h>
9: #ifndef __VXWORKS__
10: #include <sys/poll.h>
11: #include <getopt.h>
12: #endif
13: #include <errno.h>
14: #include <unistd.h>
15: #include <ctype.h>
16: #include <sys/socket.h>
17: #include <sys/types.h>
18: #include <sys/time.h>
19: #include <netinet/in.h>
20: #include <arpa/inet.h>
21: #include <vector>
22: #include <list>
23: #include <pthread.h>
24: #include <fcntl.h>
25:
26: #ifdef WIN32
27: #include <windows.h>
28: #include <process.h>
29: #endif
30:
31: #include "../common/message/SDMReady.h"
32: #include "../common/message/SDMTask.h"
33: #include "../common/message/SDMRegPM.h"
34: #include "../common/message/SDMCancelxTEDS.h"
35: #include "../common/message/SDMxTEDS.h"
36: #include "../common/message/SDMCode.h"
37: #include "../common/message/SDMReqCode.h"
38: #include "../common/message/SDMError.h"
39: #include "../common/message/SDMTaskFinished.h"
```

```

40: #include "../common/message/SDMSubreqst.h"
41: #include "../common/message/SDMCommand.h"
42: #include "../common/message/SDMReqReg.h"
43: #include "../common/message/SDMRegInfo.h"
44: #include "../common/message/SDMConsume.h"
45: #include "../common/message/SDMCancel.h"
46: #include "../common/message/SDMHeartbeat.h"
47: #include "../common/message/SDMTaskError.h"
48: #include "../common/message/SDMKill.h"
49: #include "../common/message/SDMDMLLeader.h"
50: #include "../common/message/SDMHello.h"
51: #include "../common/message/SDMAck.h"
52: #include "../common/message/SDMRegister.h"
53: #include "../common/message/SDMID.h"
54: #include "../common/Debug.h"
55: #include "../common/UDPcom.h"
56: #include "../common/checksum/checksum.h"
57: #include "../common/SubscriptionManager/SubscriptionManager.h"
58: #include "../common/MessageLogger/MessageLogger.h"
59: #include "PMProcessList.h"
60: #include "../common/task/SDMTaskResources.h"
61:
62: /*Windows uses closesocket instead of close for socket descriptors*/
63: #ifdef WIN32
64: #ifdef close
65: #undef close
66: #endif
67: #define close closesocket
68: #endif
69:
70: #define MAX_TASK_FAILURES 5 // The number of times a task will be restarted if heartbeats
timeout
71: //system returns this value if a file is not found
72: #define FILE_NOT_FOUND 32512
73: #define PORT_MSG_LENGTH 5
74: #define TM_Mode_ID 1 //ID number used in RegInfo and ReqReg messages for TM's status
message
75:
76: // Message IDs
77: const SDMMessage_ID NOTI_PERF_COUNTERS(2,13);
78: const SDMMessage_ID CMD_ENABLE_LOGGING(3,16);

```

```

79: const SDMMMessage_ID CMD_DISABLE_LOGGING(3,17);
80:
81: // Function prototypes
82: int VerifyTM(void);
83: void* Listener(void* args);
84: void RegisterPM(void);
85: void TaskLauncher(char* filename,long pid);
86: PROCESS_HANDLE LaunchTask (char *strTaskFilename, long lSdmPid);
87: void* SigHandler(void* args);
88: #ifdef WIN32
89: void* SigChldThreadFunc(void* args);
90: #endif
91: void SigChldHandler(int sig);
92: void SigIntHandler(int sig);
93: void SigTermHandler(int sig);
94: void SigAlrmHandler(int sig);
95: void SigIntHandler(int sig_num);
96: void CommandHandler(char *buf);
97: void SubreqstHandler(char *buf);
98: void DeletesubHandler(char *buf);
99: void KillHandler(char *buf);
100: void HeartbeatHandler(char *buf);
101: void* TaskHandler(void* args);
102: void PublishPerformanceCounterMessage(void);
103: void DoReset();
104: void RegisterXteds();
105: void SendRunningTasks(void);
106: void MessageSent(SDMmessage *msg);
107: void MessageReceived(SDMmessage* msg);
108: bool IsFileAvailable(const char* Filename);
109: void ErrorHandler(const char* buf);
110: void* TaskHeartbeatMonitor(void* args);
111: int SetCloseFileOnExec(int fd);
112: void RemoveCodeReceiver(const char* strFilename);
113: void DMLeaderHandler(char*);
114: double GetCurTime();
115:
116: /*Structure that represents all information needed to initiate a code transfer*/
117: struct code_rcv
118: {
119:     int fd_code;

```

```
120: #ifdef BUILD_FOR_IDS
121:     unsigned short ids_handle;
122: #endif
123:     char filename[MAX_FILENAME_SIZE];
124: };
125:
126:
127: #endif
```

File: sdm/pm/PMProcessList.cpp

```
1: #include "PMProcessList.h"
2: #include <stdio.h>
3:
4: PMProcessList::PMProcessList()
5: : m_ProcessList()
6: {}
7:
8: /*
9:  * Add a process to the list. If the new process is a restart of a failed process, the PID
10:  * will already exist in this list, the fail count will be greater than zero. In this case
11:  * don't actually add another item in the list, but update the OsPid.
12:  * Returns:
13:  *  int - The index of the added element.
14:  */
15: int PMProcessList::Add(const char* TaskName, unsigned int SdmPid, PROCESS_HANDLE OsPid)
16: {
17:     int iIndex = GetIndexOfSdmPid(SdmPid);
18:     // If the process didn't previously exist
19:     if (iIndex == -1)
20:     {
21:         PMProcess NewProc(TaskName, SdmPid, OsPid);
22:         m_ProcessList.push_back(NewProc);
23:         iIndex = m_ProcessList.size() - 1;
24:     }
25:     else if (iIndex != -1 && m_ProcessList[iIndex].GetSdmPid() == SdmPid)
26:     {
27:         // Sanity check
28:         if (m_ProcessList[iIndex].GetFailCount() == 0)
29:             printf("Warning: PMProcessList::Add()- inconsistent state, failure count shouldn't be zero.
30: \n");
31:         m_ProcessList[iIndex].SetOsPid(OsPid);
32:     }
33: #ifdef DEBUG_PM_PROCESS_LIST
34:     PrintList();
35: #endif
36:     return iIndex;
37: }
38:
```

```

39: bool PMProcessList::Remove(unsigned int Index)
40: {
41: if (Index > m_ProcessList.size())
42:     return false;
43:
44: m_ProcessList.erase(m_ProcessList.begin() + Index, m_ProcessList.begin() + Index + 1);
45: #ifdef DEBUG_PM_PROCESS_LIST
46: PrintList();
47: #endif
48: return true;
49: }
50:
51: int PMProcessList::GetIndexOfOsPid(PROCESS_HANDLE OsPid) const
52: {
53: for (unsigned int i = 0; i < m_ProcessList.size(); i++)
54: {
55:     if (m_ProcessList[i].GetOsPid() == OsPid)
56:         return i;
57: }
58: return -1;
59: }
60:
61: int PMProcessList::GetIndexOfSdmPid(unsigned int SdmPid) const
62: {
63: for (unsigned int i = 0; i < m_ProcessList.size(); i++)
64: {
65:     if (m_ProcessList[i].GetSdmPid() == SdmPid)
66:         return i;
67: }
68: return -1;
69: }
70:
71: void PMProcessList::UpdateOsPid(unsigned int SdmPid, PROCESS_HANDLE OsPid)
72: {
73: int iIndex = GetIndexOfSdmPid(SdmPid);
74: if (iIndex == -1)
75: {
76:     printf("PMProcessList::HeartbeatReceived: Invalid SdmPid (%u) \n", SdmPid);
77:     return;
78: }
79: m_ProcessList[iIndex].SetOsPid(OsPid);

```



```

80:
81: #ifdef DEBUG_PM_PROCESS_LIST
82: PrintList();
83: #endif
84: }
85:
86: void PMProcessList::HeartbeatReceived(unsigned int SdmPid)
87: {
88: int Index = GetIndexOfSdmPid(SdmPid);
89: if (Index == -1)
90: {
91:     printf("PMProcessList::HeartbeatReceived: Invalid SdmPid (%u) \n", SdmPid);
92:     return;
93: }
94:
95: m_ProcessList[Index].HeartbeatReceived();
96: }
97:
98: PMProcess& PMProcessList::operator[] (unsigned int index)
99: {
100:     return m_ProcessList[index];
101: }
102:
103: void PMProcessList::RemoveAll()
104: {
105:     m_ProcessList.clear();
106: }
107:
108: #ifdef DEBUG_PM_PROCESS_LIST
109: void PMProcessList::PrintList()
110: {
111:     printf("*****Process List (%u items)***** \n",
m_ProcessList.size());
112:     for (unsigned int i = 0; i < m_ProcessList.size(); i++)
113:     {
114:         printf("Index %u: Name = \"%s\" SdmPid = %u OsPid = %d \n", i,
m_ProcessList[i].GetName(), m_ProcessList[i].GetSdmPid(), m_ProcessList[i].GetOsPid());
115:     }
116:     printf("***** \n");
117: }
118: #endif

```

119:

File: sdm/pm/PendingTask.cpp

```
1: #include "PendingTask.h"
2: #include <string.h>
3:
4:
5: PendingTask::PendingTask() : m_strName(), m_ulSdmPid(0)
6: {
7:   m_strName[0] = '\0';
8: }
9:
10: PendingTask::PendingTask(const PendingTask& right) : m_strName(), m_ulSdmPid(0)
11: {
12:   this->m_ulSdmPid = right.m_ulSdmPid;
13:   strncpy(this->m_strName, right.m_strName, sizeof(m_strName));
14:   this->m_strName[MAX_FILENAME_SIZE-1] = '\0';
15: }
16:
17: PendingTask& PendingTask::operator=(const PendingTask& right)
18: {
19:   this->m_ulSdmPid = right.m_ulSdmPid;
20:   strncpy(this->m_strName, right.m_strName, sizeof(m_strName));
21:   this->m_strName[MAX_FILENAME_SIZE-1] = '\0';
22:   return *this;
23: }
24:
25: PendingTask::PendingTask(const char* strTaskName, unsigned long ulSdmPid)
26: : m_strName(), m_ulSdmPid(0)
27: {
28:   strncpy(m_strName, strTaskName, sizeof(m_strName));
29:   m_strName[MAX_FILENAME_SIZE-1] = '\0';
30:   m_ulSdmPid = ulSdmPid;
31: }
```

File: sdm/pm/PMProcessList.h

```
1: #ifndef _SDM_PM_PROCESS_LIST_H_
2: #define _SDM_PM_PROCESS_LIST_H_
3:
4: #include "PMProcess.h"
5: #include <vector>
6: #include <stdio.h>
7:
8: // #define DEBUG_PM_PROCESS_LIST 1
9:
10: using namespace std;
11:
12: class PMProcessList
13: {
14: public:
15:     PMProcessList();
16:
17:     int Add (const char* TaskName, unsigned int SdmPid, PROCESS_HANDLE OsPid = 0);
18:     bool Remove (unsigned int Index);
19:     size_t Size() { return (int)m_ProcessList.size(); }
20:
21:     int GetIndexOfOsPid(PROCESS_HANDLE OsPid) const;
22:     int GetIndexOfSdmPid(unsigned int SdmPid) const;
23:     void UpdateOsPid(unsigned int SdmPid, PROCESS_HANDLE OsPid);
24:     void HeartbeatReceived(unsigned int SdmPid);
25:     void RemoveAll();
26:     PMProcess& operator [] (unsigned int index);
27: #ifdef DEBUG_PM_PROCESS_LIST
28:     void PrintList();
29: #endif
30: private:
31:     vector<PMProcess> m_ProcessList;
32: };
33:
34: #endif
```

File: sdm/pm/PMProcess.cpp

```
1: #include <string.h>
2: #include <signal.h>
3: #include <sys/types.h>
4: #include <stdio.h>
5: #include "PMProcess.h"
6: #include "../common/Time/SDMTime.h"
7:
8: // Static initialization
9: bool PMProcess::m_bTimerInitialized = false;
10:
11: PMProcess::PMProcess()
12: :   m_strFilename(),   m_ulLastHeartbeat(0),   m_ulHeartbeatMissCount(0),   m_uiSdmPid(0),
   m_OsPid(0), m_uiTotalFailCount(0), m_bFailedState(false)
13: #ifdef WIN32
14: , m_bCode(false)
15: #endif
16: {}
17:
18: PMProcess::PMProcess(const char* Name, unsigned int SdmPid, PROCESS_HANDLE OsPid)
19: :   m_strFilename(),   m_ulLastHeartbeat(0),   m_ulHeartbeatMissCount(0),   m_uiSdmPid(SdmPid),
   m_OsPid(OsPid), m_uiTotalFailCount(0), m_bFailedState(false)
20: #ifdef WIN32
21: , m_bCode(false)
22: #endif
23: {
24:   strncpy(m_strFilename, Name, sizeof(m_strFilename));
25: }
26:
27: PMProcess::~PMProcess()
28: {}
29:
30: bool PMProcess::Kill() const
31: {
32: #ifndef WIN32
33:   if (kill(m_OsPid, SIGKILL) == 0)
34:     return true;
35:   return false;
36: #else
37:   return TerminateProcess(m_OsPid, 0);
```

```

38: #endif
39: }
40:
41: unsigned int PMProcess::GetCurTimeSeconds()
42: {
43:     unsigned int uiSeconds, uiUSeconds;
44:     if (!m_bTimerInitialized)
45:     {
46:         SDM_TimeInit();
47:         m_bTimerInitialized = true;
48:     }
49:     SDM_GetTime(&uiSeconds, &uiUSeconds);
50:     return uiSeconds;
51: }
52:
53: void PMProcess::HeartbeatReceived()
54: {
55:     m_ulLastHeartbeat = GetCurTimeSeconds();
56:     m_ulHeartbeatMissCount = 0;
57: }
58:
59: bool PMProcess::IsUnresponsive()
60: {
61:     // Don't try unless at least one heartbeat has been received
62:     if (m_ulLastHeartbeat == 0 && !m_bFailedState)
63:         return false;
64:
65:     unsigned int uiCurSeconds = GetCurTimeSeconds();
66:     if (uiCurSeconds < m_ulLastHeartbeat) // Sanity check
67:     {
68:         printf("Warning, current time is less than last received heartbeat. \n");
69:         return false;
70:     }
71:
72:     if (uiCurSeconds - m_ulLastHeartbeat < TASK_HEARTBEAT_TIMEOUT)
73:         return false;
74:     else
75:     {
76:         // "Fake" a heartbeat, to allow another full interval for timeout
77:         m_ulLastHeartbeat += TASK_HEARTBEAT_TIMEOUT;
78:         if (++m_ulHeartbeatMissCount >= TASK_TIMEOUT_ATTEMPTS)

```

```
79:         return true;
80: }
81: return false;
82: }
83:
84: /*
85: NOW INLINE
86: bool PMProcess::IsNormalFinish() const
87: {
88: return !m_bFailedState;
89: }
90: */
91: void PMProcess::Failure()
92: {
93: m_bFailedState = true;
94: m_uiTotalFailCount++;
95: m_ulLastHeartbeat = 0;
96: m_ulHeartbeatMissCount = 0;
97: }
98:
99: void PMProcess::Reset()
100: {
101:     m_OsPid = 0;
102:     m_bFailedState = false;
103: }
104:
105:
106:
107:
108:
```

File: sdm/pm/pm_main.cpp

```
1: //*****
2: //Process Manager Module of the SDM
3: //
4: //This source is compilable on both a Linux and a Win32 architecture. Throughout, there are
5: //compile switches as #ifdef ... #endif which conditionally add or remove code based on the build.
6: //The way by which Windows and Linux handle forking/execing processes and signal handling is
7: //very different, because of this there are different versions of a couple of function for each.
8: //*****
9: #include "pm_main.h"
10: #include "pmxTEDS.h"
11: #include "PendingTask.h"
12: #include <errno.h>
13: #include "../common/Time/SDMTime.h"
14: #include "../common/semaphore/semaphore.h"
15:
16: #ifndef WIN32
17: # include <net/if.h>
18: # include <netdb.h>
19: #endif
20:
21: #ifdef __VXWORKS__
22: #include <rtpLib.h>
23: #include <taskLib.h>
24: #include <hostLib.h>
25: #include <pipeDrv.h>
26: #include <usrFsLib.h>
27: #endif
28:
29: #ifdef BUILD_FOR_IDS
30: # include "pm_ids.h"
31: #else
32: # include "pm.h"
33: #endif
34:
35: using namespace std;
36:
37: int debug = 0;
38: bool spacewire = false;
39: bool merged = false;
```



```

40: char* workingDirectory = "./Logs";
41:
42: unsigned int total_recd = 0;    // message counter for total received for life of pm
43: unsigned int prevsec_recd = 0;  // message counter for total received previous second
44: unsigned int total_sent = 0;    // message counter for total sent for life of pm
45: unsigned int prevsec_sent = 0;  // message counter for total sent previous second
46: pthread_mutex_t perfCounterMutex = PTHREAD_MUTEX_INITIALIZER;
47:
48: SDMMMessage_ID tm_mode_msg_id;
49: unsigned char tm_mode = 0;
50: SDMComponent_ID tm_mode_source;
51: SubscriptionManager subscriptions;
52: pthread_mutex_t subscriptionsMutex = PTHREAD_MUTEX_INITIALIZER;
53:
54: unsigned long Address_PM = inet_addr("127.0.0.1");
55: bool bgTMDetected = false;    // Bool determining whether the TM has already been found
56:
57: vector<PendingTask> taskQueue;
58: pthread_mutex_t taskQueueMutex = PTHREAD_MUTEX_INITIALIZER;
59:
60: vector<code_rcv> codeReceivers; // Vector of all tasks currently doing a code transfer
61: pthread_mutex_t codeReceiversMutex = PTHREAD_MUTEX_INITIALIZER;
62:
63: PMProcessList runningTaskList;
64: pthread_mutex_t taskListMutex = PTHREAD_MUTEX_INITIALIZER;
65:
66: MessageLogger log_service;    // Message Logger utility
67:
68: const unsigned int THREAD_STACK_SIZE = 256000;
69:
70: int g_iPMListenSock = IP_SOCKET_INVALID;
71:
72: bool ackReceived = false;
73: bool registerReceived = false;
74: bool idReceived = false;
75:
76: #ifdef __uClinux__
77: SDMTaskResources g_TaskResources(SDM_LINUX26 | SDM_MICROBLAZE | SDM_MEM128);
78: #elif WIN32
79: SDMTaskResources g_TaskResources(SDM_WIN32 | SDM_INTEL | SDM_MEM1024);
80: #else

```

```

81: SDMTaskResources g_TaskResources(SDM_LINUX26 | SDM_INTEL | SDM_MEM128);
82: #endif
83: Sem g_semLaunchTask(1);
84:
85: int main(int argc, char** argv)
86: {
87: char* LOCAL_ADDR = "127.0.0.1";
88: SDMxTEDS pm_xteds;
89: SDM_TimeInit();
90:
91: bool TM_set = false;
92: //parse command line options
93: while(1)
94: {
95: #ifndef __VXWORKS__
96:     static struct option long_options[] = {
97:         {"dm", 1, 0, 'd'},
98:         {"tm", 1, 0, 't'},
99:         {"help", 0, 0, 'h'},
100:        {"debug", 1, 0, 'g'},
101:        {"spacewire", 1, 0, 's'},
102:        {"nodeid", 1, 0, 'n'},
103:        {"merged", 1, 0, 'm'},
104:        {"workingDirectory", 1, 0, 'w'},
105:        {0, 0, 0, 0} };
106:        int option_index;
107:        int option = getopt_long(argc, argv, "g:s:hn:d:t:n:mw:", long_options, &option_index);
108:    #else
109:        int option = getopt(argc, argv, "g:s:hn:d:t:n:mw:");
110:    #endif
111:
112:        if(option == -1)
113:            break; //no more options
114:        switch(option)
115:        {
116:            case 'd':
117:                //DM address no longer needed, it is acquired by the TM
118:                break;
119:            case 'g':
120:                debug = atoi(optarg);
121:                break;

```

```

122:         case 'h':
123:             printf("Usage: pm [options] \n");
124:             printf("Options: \n");
125:             //printf("--dm=<addr> -d<addr> \t \t \tSet IP address of Data Manager \n \t \t \t \t
\t<addr> should be given in dot number notation \n");
126:             printf("--debug=<level> -g<level> \t \tSet level of debug messages \n \t \t \t \t
\t0=none, 1=moderate, 2=verbose \n");
127:             printf("--tm=<addr> -t<addr> \t \t \tSet IP address of Task Manager \n \t \t \t \t
\t<addr> should be given in dot number notation \n");
128:             printf("--help -h \t \t \tDisplay this information \n");
129:             printf("--merged -m \t \t \tMerge the output of all child processes with pm's output.
Default is to log them \n");
130:             printf("--nodeid=<id> -n<id> \t \t \tSet this node's id for preferred PM scheduling
from the TM. \n");
131:             printf("--spacewire=<bool> -s<bool> \t \tEnable or disable SpaceWire mode \n");
132:             printf("--workingDirectory=<path> -w<path> \t \tLogging directory. Defaults to
./Logs' \n");
133:             return 0;
134:         case 'm':
135:             merged = true;
136:             break;
137:         case 'n':
138:         {
139:             int iNodeId = atoi(optarg);
140:             if (iNodeId < 1 || iNodeId > 255)
141:             {
142:                 debug_f(0, "Invalid nodeid specified on command-line of %d \n", iNodeId);
143:                 break;
144:             }
145:             g_TaskResources.SetPreferredPmNodeId(static_cast<unsigned short>(iNodeId));
146:             break;
147:         }
148:         case 's':
149:             spacewire = !!atoi(optarg);
150:             break;
151:         case 't':
152:             // If spacewire mode is enabled, this setting will have no effect
153:             if(strcmp(optarg,"local")==0)
154:             {
155:                 TaskManager.setAddress(inet_addr(LOCAL_ADDR));
156:                 TaskManager.setPort(PORT_TM);
157:             }

```

```

158:         else
159:         {
160:             TaskManager.setAddress(inet_addr(optarg));
161:             TaskManager.setPort(PORT_TM);
162:         }
163:         TM_set = true;
164:         break;
165:     case 'w':
166:         workingDirectory = strdup(optarg);
167:         break;
168:     case '?':
169:         break;
170:     }
171: }
172:
173: if ( spacewire )
174:     debug_f(1, "Running in SpaceWire mode \n");
175:
176: if(!TM_set && !spacewire)
177: {
178:     TaskManager.setAddress(inet_addr(LOCAL_ADDR));
179:     TaskManager.setPort(PORT_TM);
180: }
181: else if(spacewire)
182: {
183: #ifndef __VXWORKS__
184:     unsigned long addr;
185:     struct hostent *he;
186:     while ((he=gethostbyname("taskmanager.spacewire")) == NULL) {
187:         debug_f(1, "gethostname failed for taskmanager.spacewire. Retrying in 5 secs \n");
188:         sleep(5);
189:     }
190:     memcpy(&addr, he->h_addr, sizeof(addr));
191: #else
192:     int addr;
193:     while ((addr = hostGetByName("taskmanager.spacewire")) == ERROR)
194:     {
195:         debug_f(1, "gethostname failed for taskmanager.spacewire. Retrying in 5 secs \n");
196:         sleep(5);
197:     }
198: #endif

```

```

199:
200:     TaskManager.setAddress(addr);
201:     TaskManager.setPort(PORT_TM);
202:     TM_set = true;
203: }
204:
205: //output version information (including debug message style)
206: printf ("PM (Process Manager) %s \n",SDM_VERSION);
207: switch(debug)
208: {
209:     case 1:
210:         printf("PM in debug 1 (moderate). \n");
211:         break;
212:     case 2:
213:         printf("PM in debug 2 (verbose). \n");
214:         break;
215:     case 3:
216:         printf("PM in debug 3 (verbose w/message echo). \n");
217:         break;
218: }
219: pthread_attr_t threadAttr;
220: pthread_attr_init(&threadAttr);
221: pthread_attr_setstacksize(&threadAttr, THREAD_STACK_SIZE);
222: pthread_attr_setdetachstate(&threadAttr, PTHREAD_CREATE_DETACHED);
223: #ifndef WIN32
224:     //All subsequent threads block the below signals so they aren't interrupted after calling
pthread_mutex_lock()
225:     //this avoids a deadlock situation by dedicating a single thread for signal handling.
226:
227:     sigset_t signal_set;
228:     sigemptyset(&signal_set);
229:     sigaddset(&signal_set, SIGALRM);
230:     sigaddset(&signal_set, SIGINT);
231:     //sigaddset(&signal_set, SIGSEGV);
232:     sigaddset(&signal_set, SIGCHLD);
233:     pthread_sigmask(SIG_BLOCK, &signal_set, NULL);
234:
235:     pthread_t SigHandlerThread;
236:     if (0 != pthread_create(&SigHandlerThread, &threadAttr, &SigHandler, NULL))
237:     {
238:         printf("Error spawning signal handler thread. \n");

```

```

239:     return -1;
240: }
241: #else //WIN32 case
242: // Start a separate thread for SigChldHandling
243: pthread_t SigHandlerThread;
244: if (0 != pthread_create(&SigHandlerThread, &threadAttr, &SigChldThreadFunc, NULL))
245: {
246:     printf("Error spawning SigChld thread. \n");
247:     return -1;
248: }
249: signal(SIGINT, *SigIntHandler);
250: sigset(SIGALRM, SigAlrmHandler);
251: #endif
252: pthread_t HeartbeatMonitorThread;
253: if (0 != pthread_create(&HeartbeatMonitorThread, &threadAttr, &TaskHeartbeatMonitor,
NULL))
254: {
255:     printf("Error spawning heartbeat handler thread. \n");
256:     return -1;
257: }
258:
259: if (VerifyTM() == -1)
260:     return -1;
261:
262: pthread_attr_t ListenThreadAttr;
263: pthread_attr_init(&ListenThreadAttr);
264: pthread_attr_setstacksize(&ListenThreadAttr, THREAD_STACK_SIZE);
265: pthread_t ListenerThread;
266: if (0 != pthread_create(&ListenerThread, &ListenThreadAttr, &Listener, NULL))
267: {
268:     perror("Could not create Listen thread. \n");
269:     return -1;
270: }
271:
272: SDMHHello hello;
273: hello.source.setPort(PORT_PM);
274: hello.type = 'C';
275: double endTime = 0;
276: double timeOut = 5.0;
277: while(!ackReceived)
278: {

```

```

279:     if(GetCurTime() > endTime)
280:     {
281:         debug_f(1, "Sending Hello \n");
282:         hello.Send();
283:         endTime = GetCurTime() + timeOut;
284:     }
285:     usleep(10000);
286: }
287: while(!registerReceived)
288: {
289:     usleep(10000);
290: }
291: debug_f(1, "Registering xTEDS \n");
292:
293:
294:     pm_xteds.source.setSensorID(1);
295:     pm_xteds.source.setPort(PORT_PM);
296:     strcpy(pm_xteds.xTEDS, xTEDS);
297:     MessageSent(&pm_xteds);
298:     pm_xteds.Send();
299:
300: while (!idReceived)
301: {
302:     usleep(10000);
303: }
304: debug_f(1, "SDMID received \n");
305:
306:     RegisterPM();
307:
308: pthread_join(ListenerThread, NULL);
309:
310:     return 0;
311: }
312:
313:
314:
315: double GetCurTime()
316: {
317:     unsigned int seconds;
318:     unsigned int uSeconds;
319:     double curTime;

```

```

320:  SDM_GetTime(&seconds, &uSeconds);
321:  curTime = seconds + ((double)uSeconds/1000000.0);
322:  return curTime;
323: }
324:
325:
326: //////////////////////////////////////
327: //
328: //          START SIGNAL HANDLING FUNCTIONS
329: //
330: //////////////////////////////////////
331: #ifdef WIN32
332: void* SigChldThreadFunc(void* args)
333: {
334:     while (1)
335:     {
336:         SigChldHandler(SIGCHLD);
337:         usleep(10000);
338:     }
339: }
340: #endif
341: #ifndef WIN32
342: void* SigHandler(void* args)
343: {
344:     sigset_t signal_set;
345:     sigset_t pending_set;
346:     int sig;
347:     int sig_pending;
348:
349:     sigemptyset(&signal_set);
350:     sigaddset(&signal_set, SIGINT);
351:     sigaddset(&signal_set, SIGALRM);
352:     sigaddset(&signal_set, SIGSEGV);
353:     sigaddset(&signal_set, SIGCHLD);
354:
355:     while (1)
356:     {
357:         sig_pending = sigpending(&pending_set);
358:
359:         if (sig_pending == 0 && (sigismember(&pending_set, SIGINT) ||
360:             sigismember(&pending_set, SIGALRM) || sigismember(&pending_set, SIGSEGV) ||

```



```

361:    sigismember(&pending_set, SIGCHLD)))
362:    {
363:        sigwait(&signal_set, &sig);
364:
365:        switch (sig)
366:        {
367:            case SIGINT:
368:                SigIntHandler(SIGINT);
369:                break;
370:            case SIGALRM:
371:                SigAlrmHandler(SIGALRM);
372:                break;
373:            case SIGSEGV:
374:                SigIntHandler(SIGSEGV);
375:                break;
376:            case SIGCHLD:
377:                SigChldHandler(SIGCHLD);
378:                break;
379:        }
380:    }
381:    else
382:    {
383:        // Periodically "fake" a SIGCHLD because multiple SIGCHLDs simultaneously may
384:        // overwrite a previous one
385:        SigChldHandler(SIGCHLD);
386:        usleep(500000);
387:    }
388: }
389: return NULL;
390: }
391: #endif
392: /*
393:  Signal handler routine that handles SIGCHLD, which is raise any time a forked child process has
394:  exited. The PM will keep a list of
395:  running tasks and remove the PID of the finished task, and sent an SDMTaskFinished message to
396:  the TM. If the TM's mode changes to
397:  the mode signifying a DM failure, the SDMTaskFinished message is not sent, and this function
398:  just collects the return value of the
399:  child process.
400:  INPUTS:
401:      int sig -- The signal number (SIGCHLD)
402:  RETURNS:

```

```

400:     void -- nothing
401: */
402: void SigChldHandler(int sig)
403: {
404:     //
405:     // See if there are any queued tasks to be started
406:     pthread_mutex_lock(&taskQueueMutex);
407:     size_t uiTaskQueueSize = taskQueue.size();
408:     pthread_mutex_unlock(&taskQueueMutex);
409:     if (uiTaskQueueSize > 0)
410:     {
411:         debug_f(3, "taskQueue contains %d items. \n", uiTaskQueueSize);
412:         pthread_mutex_lock(&taskQueueMutex);
413:         PendingTask launchTask = taskQueue[0];
414:         taskQueue.erase(taskQueue.begin(), taskQueue.begin() + 1);
415:         pthread_mutex_unlock(&taskQueueMutex);
416:         LaunchTask(launchTask.GetName(), launchTask.GetPid());
417:     }
418:     //
419:     // See if any finished tasks need to be retrieved
420:     int iStatus = 0;
421:     PROCESS_HANDLE iPid = 0;
422:     SDMTaskFinished msgFinished;
423:     SDMCancelxTEDS msgCancel;
424: #ifdef WIN32
425:     bool bFoundFinished = false;
426:     unsigned long ulStatus = 0;
427:     for (unsigned int i = 0; i < runningTaskList.Size(); i++)
428:     {
429:         iPid = runningTaskList[i].GetOsPid();
430:         GetExitCodeProcess(iPid, &ulStatus);
431:         if (ulStatus == STILL_ACTIVE)
432:             continue;
433:         else
434:         {
435:             bFoundFinished = true;
436:             break;
437:         }
438:     }
439:     // If no finished processes were found, return
440:     if (!bFoundFinished)

```

```

441:         return;
442:     iStatus = static_cast<int>(ulStatus);
443: #else
444:     // See what child process has finished execution
445:     iPid = waitpid(-1, &iStatus, WNOHANG);
446:     debug_f(4, "iPid is %d \n", iPid);
447:     if (iPid == 0 || iPid == -1)
448:         return;
449: #endif
450:     debug_f(3, "Finished pid is %p \n", iPid);
451:
452:     // Find the finished process in the runningTaskList list and removed it, also send an
SDMTaskFinished message to the TM*/
453:     pthread_mutex_lock(&taskListMutex);
454:     int TaskIndex = runningTaskList.GetIndexOfOsPid(iPid);
455:     if (TaskIndex == -1)
456:     {
457:         printf("Could not find a running task with pid %p \n", iPid);
458:         pthread_mutex_unlock(&taskListMutex);
459:         return;
460:     }
461:     // If this was not a normal finish (i.e. the task was unresponsive and killed), restart the task
462:     if (runningTaskList[TaskIndex].IsNormalFinish() == false)
463:     {
464:         // Reset the fail state
465:         runningTaskList[TaskIndex].Reset();
466:
467:         SDMTask msgTask;
468:         // Restart the task, reusing the same SDM PID
469:         msgTask.pid = runningTaskList[TaskIndex].GetSdmPid();
470:         strncpy(msgTask.filename, runningTaskList[TaskIndex].GetName(),
sizeof(msgTask.filename));
471:         SDMComponent_ID PmId;
472:         PmId.setAddress(Address_PM);
473:         PmId.setPort(PORT_PM);
474:         msgTask.Send(PmId);
475:
476:         pthread_mutex_unlock(&taskListMutex);
477:         return;
478:     }
479:     // Otherwise, normal finish, inform the TM and DM

```

```

480:  debug_f(1,"Finished %s task with return code %d. \n", runningTaskList[TaskIndex].GetName(),
iStatus);
481:  if (tm_mode != MODE_HARD_RESET || tm_mode != MODE_SOFT_RESET)
482:  {
483:      printf("Informing TM and DM of termination... \n");
484:      // Inform the TM of the finished task
485:      strncpy(msgFinished.filename, runningTaskList[TaskIndex].GetName(),
MAX_FILENAME_SIZE);
486:      msgFinished.result = iStatus;
487:      msgFinished.pid = runningTaskList[TaskIndex].GetSdmPid();
488:      msgFinished.source.setAddress(Address_PM);
489:      msgFinished.source.setPort(PORT_PM);
490:      msgFinished.Send(); //Send the return code to the TM
491:      MessageSent(&msgFinished);
492:
493:      // Ensure that the xTEDS for this task is cancelled
494:      msgCancel.source.setPort(PORT_PM);
495:      msgCancel.source.setAddress(Address_PM);
496:      msgCancel.source.setSensorID(runningTaskList[TaskIndex].GetSdmPid());
497:      msgCancel.Send();
498:      MessageSent(&msgCancel);
499:  }
500:  runningTaskList.Remove(TaskIndex);
501:  pthread_mutex_unlock(&taskListMutex);
502: }
503:
504: /*
505:  The signal_handler routine is used to respond to a Ctrl-C
506:  INPUTS:
507:      signal - the signal that the routine caught
508:  */
509: void SigIntHandler(int signal)
510: {
511:     SDMCancelxTEDS cancel;
512:     SDMCancel cancel_msg;
513:     // Shutdown the ProcessManager
514:     if (signal == SIGINT)
515:     {
516:         // Cancel the ProcessManager from the DataManager
517:         cancel.source.setPort(PORT_PM);
518:         cancel.source.setAddress(Address_PM);

```

```

519:     cancel.Send();
520:     MessageSent(&cancel);
521:
522:     // For each task, terminate their execution and cancel their xTEDS (if registered)
523:     pthread_mutex_lock(&taskListMutex);
524:     for (unsigned int i = 0; i < runningTaskList.Size(); i++)
525:     {
526:         // Terminate the process
527:         runningTaskList[i].Kill();
528:
529:         // Cancel this process's xTEDS
530:         cancel.source.setSensorID(runningTaskList[i].GetSdmPid());
531:         cancel.Send();
532:         MessageSent(&cancel);
533:         debug_f(3,"Sending   SDMCancelxTEDS   for   SDM   pid   %d   \n",
runningTaskList[i].GetSdmPid());
534:     }
535:     pthread_mutex_unlock(&taskListMutex);
536:
537:     // Cancel the subscription to the TaskManager's mode message
538:     CancelReqRegSubs();
539:     // Terminate process
540:     exit(EXIT_SUCCESS);
541: }
542: if (signal == SIGSEGV)
543: {
544:     cancel.source.setPort(PORT_PM);
545:     cancel.source.setAddress(Address_PM);
546:     cancel.Send(); //inform DM of the segmentation fault
547:     MessageSent(&cancel);
548:     printf("SIGSEGV \n");
549:     return; //continue normal operation
550: }
551: }
552:
553: void SigAlrmHandler(int sig)
554: {
555:     if (sig == SIGALRM)
556:     {
557:         PublishPerformanceCounterMessage();
558:         pthread_mutex_lock(&perfCounterMutex);

```

```

559:     prevsec_recd = prevsec_sent = 0;
560:     pthread_mutex_unlock(&perfCounterMutex);
561: }
562: }
563:
564: //////////////////////////////////////
565: //
566: //             END SIGNAL HANDLING FUNCTIONS
567: //
568: //////////////////////////////////////
569:
570: void* TaskHeartbeatMonitor(void* args)
571: {
572:     //SDMCancelxTEDS msgCancel;
573:
574:     while (1)
575:     {
576:         pthread_mutex_lock(&taskListMutex);
577:         for (unsigned int i = 0; i < runningTaskList.Size(); i++)
578:         {
579:             if (runningTaskList[i].IsUnresponsive() && !runningTaskList[i].IsFailed())
580:             {
581:                 debug_f(0, " \"%s \" (SdmPid=%u, OsPid=%p) is unresponsive; restarting... \n",
runningTaskList[i].GetName(), runningTaskList[i].GetSdmPid(), runningTaskList[i].GetOsPid());
582:
583:                 // Kill the task
584:                 runningTaskList[i].Kill();
585:
586:                 if (runningTaskList[i].GetFailCount() >= MAX_TASK_FAILURES)
587:                 {
588:                     debug_f(0, "Warning: Task \"%s \" is faulty and won't be restarted. \n",
runningTaskList[i].GetName());
589:                     runningTaskList.Remove(i);
590:                     continue;
591:                     //TODO: Send SDMTTaskFinished??
592:                 }
593:                 // Send SDMTTaskFinished to DM. This tells the DM to cancel it without
594:                 // Cancel the task and restart it
595:                 SDMTTaskError msgError;
596:                 msgError.source.setAddress(Address_PM);
597:                 msgError.source.setPort(PORT_PM);

```

```

598:         msgError.source.setSensorID(runningTaskList[i].GetSdmPid());
599:         msgError.pid = runningTaskList[i].GetSdmPid();
600:         msgError.Send();
601:
602:         // The task list entry is kept, but a reset will allow it to start over
603:         // its heartbeat status and set its failure state
604:         runningTaskList[i].Failure();
605:
606:         // Note: This task will be restarted when it is wait()ed on by the
607:         // SIGCHLD handler.
608:     }
609: }
610: pthread_mutex_unlock(&taskListMutex);
611: sleep(2);
612: }
613: return NULL;
614: }
615:
616: /*
617:  PublishPerformanceCounterMessage sends out the current values on the message performance
618:  counters to all subscribers of the message.
619:  INPUTS:
620:      None.
621:  RETURNS:
622:      None.
623: */
624: void PublishPerformanceCounterMessage()
625: {
626:     char msg[16];
627:     pthread_mutex_lock(&perfCounterMutex);
628:     PUT_UINT(&msg[0], total_recd);
629:     PUT_UINT(&msg[4], prevsec_recd);
630:     PUT_UINT(&msg[8], total_sent);
631:     PUT_UINT(&msg[12], prevsec_sent);
632:     pthread_mutex_unlock(&perfCounterMutex);
633:
634:     pthread_mutex_lock(&subscriptionsMutex);
635:     if (subscriptions.Publish(NOTI_PERF_COUNTERS, msg, 16))
636:     {
637:         MessageSent(subscriptions.GetLastPublished());

```

```

638:     debug_f(3,"Publishing Message_Count \n");
639: }
640: pthread_mutex_unlock(&subscriptionsMutex);
641: }
642: /*
643:  VerifyTM finds and verifies that there is a TaskManager running.  If there is a TM running,
644:  the TM will return the DataManager's address in the SDMReady message.  If there is not a
645:  TM running, the PM will just spin in this loop until it gets a response.
646:  INPUTS:
647:      None.
648:  RETURN VALUE:
649:      None.
650: */
651: int VerifyTM(void)
652: {
653:     SDMReady msgReady;
654:     char buf[BUFSIZE];
655:     int sock = UDPpassive(PORT_PM);
656:     if (sock == IP_SOCK_INVALID)
657:     {
658:         printf("%s() could not bind port. \n", __FUNCTION__);
659:         return -1;
660:     }
661:
662:     msgReady.destination.setPort(PORT_PM);
663:     msgReady.destination.setAddress(TaskManager.getAddress());
664:     printf("Searching for TM .");
665:     //fflush(NULL);
666:     do {
667:         total_sent++;
668:         prevsec_sent++;
669:         msgReady.Send();
670:         putchar('.');
671:         //fflush(NULL);
672:     } while (! UDPAvail(sock, 250));
673:
674:     UDPrecv (sock, buf, BUFSIZE);
675:
676:     total_rcd++;
677:     prevsec_rcd++;
678:     msgReady.Unmarshal(buf);

```



```

679: // Get the actual IP address of this PM node
680: Address_PM = msgReady.destination.getAddress();
681: // Get the DM address
682: DataManager = msgReady.source;
683: UDPclose (sock);
684: printf("TM found \n");
685: debug_f(3,"Setting DM address to 0x%lx \n", DataManager.getAddress());
686: return 0;
687: }
688:
689: /*
690: RegisterPM registers this PM node with the TaskManager according to its resource type.
691: INPUTS:
692:     None.
693: RETURN VALUE:
694:     None.
695: */
696: void RegisterPM(void)
697: {
698:     SDMRegPM msgReg;
699:     msgReg.resources = g_TaskResources.GetUShort();
700:     msgReg.source.setAddress(Address_PM);
701:     msgReg.source.setPort(PORT_PM);
702:     msgReg.Send();
703:     MessageSent(&msgReg);
704: }
705:
706: /*
707: * Start a task. Setup the command-line arguments for the task and start it. Information
708: * about the task is kept in the running task list.
709: * INPUTS:
710: *     strTaskFilename - the file to be executed
711: *     lSdmPid - SDM assigned process id
712: * RETURN VALUE:
713: *     the return value is the process id of the executing task
714: */
715: PROCESS_HANDLE LaunchTask (char *strTaskFilename, long lSdmPid)
716: {
717:     PROCESS_HANDLE AppPid;    // pid of the launched application
718:     struct in_addr addrDm;    // Structure for DataManager's address
719:     struct in_addr addrTm;    // Structure for TaskManager's address

```

```

720: char strDmAddress[16];    // Char array of DataManager's address in number-dot notation
721: char strTmAddress[16];    // Char array of TaskManager's address in number-dot notation
722: char strPid[8];           // String version of the lSdmPid
723:
724: // Set the TM and DM addresses in number-dot notation to pass as command line args to the task
being launched
725: addrTm.s_addr = TaskManager.getAddress();
726: addrDm.s_addr = DataManager.getAddress();
727: strncpy(strTmAddress, inet_ntoa(addrTm), sizeof(strTmAddress));
728: strncpy(strDmAddress, inet_ntoa(addrDm), sizeof(strDmAddress));
729: sprintf(strPid, "%ld", lSdmPid);    // SDM process identifier in string format
730:
731: debug_f(2,"Launching %s task \n", strTaskFilename);
732: //
733: // Assure that the file exists
734: if (!IsFileAvailable(strTaskFilename))
735: {
736:     printf("Fatal error (LaunchTask): File does not exist for %s \n", strTaskFilename);
737:     return 0;
738: }
739: //
740: // Add the new task to the runningTaskList list
741: // Must be added before the task is started
742: pthread_mutex_lock(&taskListMutex);
743: int iTaskIndex = runningTaskList.Add(strTaskFilename, lSdmPid);
744: pthread_mutex_unlock(&taskListMutex);
745:
746: #ifdef WIN32
747: //
748: // ***** WINDOWS *****
749: //
750: // Perform win32 specific fork/exec like operations
751: PROCESS_INFORMATION proc_info;    // Structure for process information needed by
CreateProcess
752:
753: STARTUPINFO start_info;           // Structure for process startup info needed by CreateProcess
754: memset(&start_info,0,sizeof(STARTUPINFO));
755: //
756: // Setup the task arguments to be passed to the exec function
757: char strCommand[256];
758: snprintf(strCommand, sizeof(strCommand), "%s %s %s %s", strTaskFilename, strTmAddress,
strDmAddress, strPid);

```

```

759:
760:  HANDLE log;
761:
762:  CreateDirectory(workingDirectory,NULL);
763:  if ( !merged )
764:  {
765:      char name[256];
766:      name[0] = '\0';
767:      strcat_s(name,sizeof(name),"Logs \\\");
768:      char* cp = strstr(strTaskFilename, ".exe");
769:      if ( cp == NULL )
770:          strcat_s(name,sizeof(name),strTaskFilename);
771:      else
772:          strncat_s(name,sizeof(name),strTaskFilename,cp-strTaskFilename);
773:      strcat_s(name,sizeof(name), ".log");
774:      //sprintf_s(name,sizeof(name), "Logs \\\%s.log", strTaskFilename);
775:      SECURITY_ATTRIBUTES security;
776:      security.nLength = sizeof(security);
777:      security.bInheritHandle = TRUE;
778:      security.lpSecurityDescriptor = NULL;
779:      log =
CreateFile(name,GENERIC_WRITE,FILE_SHARE_READ|FILE_SHARE_WRITE,&security,
780:  CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL|FILE_FLAG_NO_BUFFERING|FILE_FLAG
_WRITE_THROUGH,NULL);
781:      start_info.dwFlags = STARTF_USESTDHANDLES;
782:      start_info.hStdOutput = log;
783:  }
784:  if (CreateProcess(strTaskFilename, strCommand, NULL, NULL, TRUE, 0, NULL,
workingDirectory, &start_info, &proc_info))
785:  {
786:      debug_f(3,"Process successfully created. \n");
787:      // Get the process handle
788:      AppPid = proc_info.hProcess;
789:      // Update the PID
790:      pthread_mutex_lock(&taskListMutex);
791:      runningTaskList.UpdateOsPid(lSdmPid, AppPid);
792:      pthread_mutex_unlock(&taskListMutex);
793:  }
794:  else
795:  {
796:      printf("Error starting task %s! (%d) \n", strTaskFilename, GetLastError());

```

```

797:
798:     pthread_mutex_lock(&taskListMutex);
799:     runningTaskList.Remove(iTaskIndex);
800:     pthread_mutex_unlock(&taskListMutex);
801:
802:     return NULL;
803: }
804: CloseHandle(log);
805: #elif defined(__VXWORKS__)
806:     //
807:     // ***** VXWORKS *****
808:     //
809:
810:     // Setup the task arguments to be passed to the exec function
811:     char strProgName[MAX_FILENAME_SIZE+10];
812:     strcpy(strProgName, "/ram0/");
813:     strcat(strTaskFilename, ".vxe");
814:     strcat(strProgName, strTaskFilename);
815:
816:     const char* taskArgs[5];
817:     const char* envVars[2];
818:     char libPath[100];
819:     strcpy(libPath, "LD_LIBRARY_PATH=");
820:     strcpy(&libPath[16], getenv("LD_LIBRARY_PATH"));
821:     taskArgs[0] = strProgName;
822:     taskArgs[1] = strTmAddress;
823:     taskArgs[2] = strDmAddress;
824:     taskArgs[3] = strPid;
825:     taskArgs[4] = NULL;
826:
827:     envVars[0] = libPath;
828:     envVars[1] = NULL;
829:
830:     g_semLaunchTask.Wait();
831:     AppPid = rtpSpawn(taskArgs[0], taskArgs, envVars, 100, 0x20000, 0, VX_FP_TASK);
832:
833:     if (AppPid < 0)
834:     {
835:         printf("Error spawning child rtp: %s \n", taskArgs[0]);
836:
837:         pthread_mutex_lock(&taskListMutex);

```

```

838:     runningTaskList.Remove(iTaskIndex);
839:     pthread_mutex_unlock(&taskListMutex);
840:
841:     g_semLaunchTask.Signal();
842:     return -1;
843: }
844:
845: // Update the PID
846: g_semLaunchTask.Signal();
847: pthread_mutex_lock(&taskListMutex);
848: runningTaskList.UpdateOsPid(lSdmPid, AppPid);
849: pthread_mutex_unlock(&taskListMutex);
850: #else
851: //
852: // ***** LINUX *****
853: //
854:
855: // Setup the task arguments to be passed to the exec function
856: char * taskArgs[5];
857: taskArgs[0] = strTaskFilename;
858: taskArgs[1] = strTmAddress;
859: taskArgs[2] = strDmAddress;
860: taskArgs[3] = strPid;
861: taskArgs[4] = NULL;
862: //
863: // Add the "./" for Linux to start the task
864: char strProgName[MAX_FILENAME_SIZE+2]; //Task name prepended with "./" for exec
865: sprintf (strProgName, "./%s", strTaskFilename);
866: //
867: // Fork a child
868: g_semLaunchTask.Wait();
869: AppPid = fork();
870: if (AppPid < 0)
871: {
872:     printf("Error forking child process.. \n");
873:
874:     pthread_mutex_lock(&taskListMutex);
875:     runningTaskList.Remove(iTaskIndex);
876:     pthread_mutex_unlock(&taskListMutex);
877:
878:     g_semLaunchTask.Signal();

```

```

879:     return -1;
880: }
881: else if (AppPid == 0)    // Child process
882: {
883: #define EXEC_RETRIES (5)
884:
885:     for(int iExecTry=0; iExecTry < EXEC_RETRIES; iExecTry++)
886:     {
887:         if (execvp(strProgName, taskArgs) < 0)
888:         {
889:             perror("Error on exec:");
890:             sleep(5);
891:         }
892:     }
893:
894:     // Absolutely must exit if exec fails
895:     _exit(0);
896: }
897: else        // Parent process
898: {
899:     // Update the PID
900:     g_semLaunchTask.Signal();
901:     pthread_mutex_lock(&taskListMutex);
902:     runningTaskList.UpdateOsPid(lSdmPid, AppPid);
903:     pthread_mutex_unlock(&taskListMutex);
904: }
905: #endif
906: return AppPid;
907: }
908:
909: /*
910:  Set the flags on "fd" to auto-close the file on an exec. This prevents any tasks from holding
911:  onto the PM's files if the PM fails and needs to restart.
912:  */
913: int SetCloseFileOnExec(int fd)
914: {
915:     return fcntl(fd, F_SETFD, FD_CLOEXEC);
916: }
917: /*
918:  Find an available port to give to an application.
919:  */

```

```

920: long FindAvailablePort()
921: {
922:     static int CurPort = PORT_APP_START;
923:     int iSock;
924:     while((iSock = UDPpassive(CurPort)) == IP_SOCK_INVALID)
925:     {
926:         CurPort++;
927:         if (CurPort >= 65534)
928:             CurPort = PORT_APP_START;
929:     }
930:
931:     UDPclose(iSock);
932:     long lReturnPort = CurPort++;
933:     return lReturnPort;
934: }
935:
936: void* Listener(void* args)
937: {
938:     char buf[BUFSIZE];
939:     int result;
940:     SDMTask task;
941:     char port_msg[PORT_MSG_LENGTH];
942:     long cur_port = PORT_APP_START;
943:     port_msg[0] = SDM_ReqPort;
944:     g_iPMListenSock = UDPpassive(PORT_PM);
945:
946:     if (g_iPMListenSock == IP_SOCK_INVALID)
947:     {
948:         printf("Could not bind PORT_PM \n");
949:         return NULL;
950:     }
951:     if (SetCloseFileOnExec(g_iPMListenSock) == -1)
952:     {
953:         printf("Error changing file flags for the listen port. \n");
954:         return NULL;
955:     }
956:
957:     printf("Waiting for tasks... \n");
958:
959:     ReqRegSubs();
960:

```

```

961: pthread_attr_t taskThreadAttr;
962: pthread_attr_init(&taskThreadAttr);
963: pthread_attr_setdetachstate(&taskThreadAttr, PTHREAD_CREATE_DETACHED);
964: pthread_attr_setstacksize(&taskThreadAttr, THREAD_STACK_SIZE);
965: pthread_t taskThread;
966:
967: while(1)
968: {
969:     result = UDPserv_recv(g_iPMListenSock,buf,BUFSIZE);
970:     pthread_mutex_lock(&perfCounterMutex);
971:     total_recd++;
972:     prevsec_recd++;
973:     pthread_mutex_unlock(&perfCounterMutex);
974:
975:     if (result > 0)
976:     {
977:         switch(buf[0])
978:         {
979:             case SDM_ACK:
980:                 ackReceived = true;
981:                 break;
982:             case SDM_Register:
983:                 registerReceived = true;
984:                 break;
985:             case SDM_ID:
986:                 idReceived = true;
987:                 break;
988:             case SDM_Task:
989:                 // Start a separate thread if code transfer is needed
990:                 if (0 != pthread_create(&taskThread, &taskThreadAttr, &TaskHandler,
memcpy(new char[result], buf, result)))
991:                 {
992:                     perror("Error spawning TaskHandler thread. \n");
993:                 }
994:                 break;
995:             case SDM_Kill:
996:                 KillHandler(buf);
997:                 break;
998:             case SDM_ReqPort:
999:                 cur_port = FindAvailablePort();
1000:                 PUT_LONG(port_msg+1, cur_port);

```



```

1001:         UDPServ_reply(g_iPMListenSock, port_msg, PORT_MSG_LENGTH);
1002:         break;
1003:     case SDM_Code:
1004:         HandleCodeMessage(buf);
1005:         break;
1006:     case SDM_Subreqst:
1007:         SubreqstHandler(buf);
1008:         break;
1009:     case SDM_Deletesub:
1010:         DeletesubHandler(buf);
1011:         break;
1012:     case SDM_Command:
1013:         CommandHandler(buf);
1014:         break;
1015:     case SDM_RegInfo:
1016:         RegInfoHandler(buf);
1017:         break;
1018:     case SDM_Data:
1019:         DataHandler(buf);
1020:         break;
1021:     case SDM_Heartbeat:
1022:         HeartbeatHandler(buf);
1023:         break;
1024:     case SDM_Error:
1025:         ErrorHandler(buf);
1026:         break;
1027:     case SDM_DMLeader:
1028:         DMLeaderHandler(buf);
1029:         break;
1030:     default:
1031:         debug_f(0,"Unexpected Message Type:%c \n",buf[0]);
1032:         break;
1033:     }
1034: }
1035: }
1036: return NULL;
1037: }
1038: /*

```

1039: DoReset basically sets the ProcessManager's state to the start state, before any tasks were started.

1040: This happens when the DataManager fails and a new DM is elected. All applications need to have the

```

1041:  DM address updated, so they are killed and restarted if needed.
1042:  INPUTS:
1043:      None.
1044:  RETURN VALUE:
1045:      None.
1046: */
1047: void DoReset()
1048: {
1049:     // Treat this cancellation as a task error message.
1050:     // This prevents the task from being posted as "inactive", which
1051:     // would cause it to get reposted as other tasks start up and
1052:     // perform requests
1053:     SDMCancelxTEDS msgCancelxTEDS;
1054:     SDMTaskError msgTaskError;
1055:
1056:     // Kill all running tasks
1057:     pthread_mutex_lock(&taskListMutex);
1058:     for (unsigned int i = 0; i < runningTaskList.Size(); i++)
1059:     {
1060:         // For now, this is a hard reset, meaning all running tasks are killed
1061:         debug_f(3,"Killing      %s      (%p)      \n",      runningTaskList[i].GetName(),
runningTaskList[i].GetOsPid());
1062:         runningTaskList[i].Kill();
1063:
1064:         msgTaskError.source.setPort(PORT_PM);
1065:         msgTaskError.source.setAddress(Address_PM);
1066:         msgTaskError.source.setSensorID(runningTaskList[i].GetSdmPid());
1067:         msgTaskError.pid = runningTaskList[i].GetSdmPid();
1068:         msgTaskError.Send();
1069:     }
1070:
1071:     runningTaskList.RemoveAll();
1072:     pthread_mutex_unlock(&taskListMutex);
1073:
1074:     // Clear the PM's subscription list
1075:     pthread_mutex_lock(&subscriptionsMutex);
1076:     subscriptions.ClearAllSubscriptions();
1077:     pthread_mutex_unlock(&subscriptionsMutex);
1078: }
1079:
1080: void RegisterXteds()

```

```

1081: {
1082: // Re-register the PM with the DM
1083: debug_f(3,"Reregistering the PM's xTEDS \n");
1084: SDMxTEDS msgxTEDS;
1085: msgxTEDS.source.setSensorID(1);
1086: msgxTEDS.source.setAddress(Address_PM);
1087: msgxTEDS.source.setPort(PORT_PM);
1088: strcpy(msgxTEDS.xTEDS, xTEDS);
1089: msgxTEDS.Send();
1090: MessageSent(&msgxTEDS);
1091: }
1092:
1093: void ErrorHandler(const char* buf)
1094: {
1095:   SDMError msgError;
1096:   long lMsgSize;
1097:   if ((lMsgSize=msgError.Unmarshal(buf)) < 0)
1098:   {
1099:     printf("Invalid SDMError message received. \n");
1100:     return;
1101:   }
1102:   MessageReceived(&msgError);
1103:
1104:   debug_f(1, "Error message received. \n");
1105:
1106:   unsigned int i = 0;
1107:   pthread_mutex_lock(&codeReceiversMutex);
1108:   size_t uiSize = codeReceivers.size();
1109:   for (i = 0; i < uiSize; i++)
1110:   {
1111:     if (!strcmp(codeReceivers[i].filename, msgError.error_msg))
1112:     {
1113:       if (write(codeReceivers[i].fd_code, buf, lMsgSize) < 0)
1114:       {
1115:         printf("Error writing error message to code consumer \n");
1116:       }
1117:     }
1118:     else
1119:     {
1120:       break;
1121:     }
1122:   }
1123:   if (i == uiSize)

```

```

1122:     printf("Recieved error message that didn't match any code consumer \n");
1123: pthread_mutex_unlock(&codeReceiversMutex);
1124: }
1125:
1126: /*
1127:  * Handle a SDMTask message from the Task Manager. If the executable for the task doesn't
  exist, a code
1128:  * transfer will be performed.
1129:  *
1130:  */
1131: void* TaskHandler(void* args)
1132: {
1133:     char* msgBuf = (char*) args;
1134:     SDMTask msgTask;
1135:     if (msgTask.Unmarshal(msgBuf) < 0)
1136:     {
1137:         printf("Invalid SDMTask message. \n");
1138:         delete [] msgBuf;
1139:         return NULL;
1140:     }
1141: #ifdef WIN32
1142:     char* cp = strstr(msgTask.filename, ".exe");
1143:     if ( cp == NULL )
1144:     {
1145:         strcat_s(msgTask.filename, sizeof(msgTask.filename), ".exe");
1146:     }
1147: #endif
1148:     MessageReceived(&msgTask);
1149:
1150:     // Check to see if code file exists
1151:     if (!IsFileAvailable(msgTask.filename))
1152:     {
1153: #ifndef __VXWORKS__
1154:         int iaCodeFd[2];    // Pipe descriptors for transferring code
1155:         //
1156:         // Setup pipes and perform a code transfer
1157:         if (pipe(iaCodeFd) != 0)
1158:         {
1159:             perror("Error calling pipe(): ");
1160:             delete [] msgBuf;
1161:             return NULL;

```

```

1162:     }
1163: #else
1164:     int iaCodeFd[2];
1165:     char pipeName[MAX_FILENAME_SIZE];
1166:     strcpy(pipeName, "/pipe/");
1167:     strcat(pipeName, msgTask.filename);
1168:     if(pipeDevCreate(pipeName, 5, BUFSIZE) != 0)
1169:     {
1170:         printf("ERROR creating pipe for code transfer: %i %s \n", errno, strerror( errno ));
1171:     }
1172:     iaCodeFd[0] = open(pipeName, O_RDONLY);
1173:     iaCodeFd[1] = open(pipeName, O_WRONLY);
1174: #endif
1175:     //
1176:     // Add a record for this code receiver thread
1177:     code_rcv receiverRecord;
1178:     strncpy(receiverRecord.filename, msgTask.filename, sizeof(receiverRecord.filename));
1179:     receiverRecord.filename[sizeof(receiverRecord.filename)-1] = '\0';
1180: #ifndef __VXWORKS__
1181:     receiverRecord.fd_code = iaCodeFd[1];    // Write end
1182: #else
1183:     receiverRecord.fd_code = iaCodeFd[1];    // Write end
1184: #endif
1185:
1186:     pthread_mutex_lock(&codeReceiversMutex);
1187:     codeReceivers.push_back(receiverRecord);
1188:     pthread_mutex_unlock(&codeReceiversMutex);
1189:     //
1190:     // Request the code
1191:     debug_f(0, "Sending request for code of %s, version %d, pid %lu \n",
1192:         msgTask.filename, msgTask.version, msgTask.pid);
1193: #ifndef __VXWORKS__
1194:     bool bSuccess = GetCode(msgTask.filename, msgTask.pid, msgTask.version, iaCodeFd[0]
/*Read end*/);
1195:     close(iaCodeFd[0]);
1196: #else
1197:     bool bSuccess = GetCode(msgTask.filename, msgTask.pid, msgTask.version, iaCodeFd[0]
/*Read end*/);
1198:     close(iaCodeFd[0]);
1199:     close(iaCodeFd[1]);
1200:     pipeDevDelete(pipeName, 1);

```

```

1201: #endif
1202:
1203:     if (bSuccess)    // If code transfer was a success
1204:     {
1205:         // Close the other end of the code transfer pipes
1206:         debug_f(2, "Code received for %, queueing task... \n", msgTask.filename);
1207:     }
1208:     else    // If code transfer failed
1209:     {
1210:         debug_f(0, "Could not retrieve %s from the SDM file system. \n", msgTask.filename);
1211:
1212:         // Inform the TM of the failure to begin
1213:         SDMTaskFinished msgFinished;
1214:         strncpy(msgFinished.filename, msgTask.filename, MAX_FILENAME_SIZE);
1215:         msgFinished.result = -1;
1216:         msgFinished.pid = msgTask.pid;
1217:         msgFinished.source.setAddress(Address_PM);
1218:         msgFinished.source.setPort(PORT_PM);
1219:         msgFinished.Send();
1220:         MessageSent(&msgFinished);
1221:
1222:         delete [] msgBuf;
1223:         return NULL;
1224:     }
1225: }
1226: //
1227: // Add the task to the task queue to be started
1228: // This fixes a uClinux problem where the thread that forks a process must
1229: // be the thread that performs a wait()/waitpid() (the signal handler thread)
1230: PendingTask newTask(msgTask.filename, msgTask.pid);
1231: pthread_mutex_lock(&taskQueueMutex);
1232: taskQueue.push_back(newTask);
1233: pthread_mutex_unlock(&taskQueueMutex);
1234:
1235: delete [] msgBuf;
1236: return NULL;
1237: }
1238:
1239: void KillHandler(char *buf)
1240: {
1241:     SDMKill msgKill;

```

```

1242: if (msgKill.Unmarshal(buf) == SDM_INVALID_MESSAGE)
1243: {
1244:     printf("Invalid kill message. \n");
1245:     return;
1246: }
1247: MessageReceived(&msgKill);
1248: debug_f(1, "Received kill command for pid %ld \n", msgKill.PID);
1249:
1250: // Find and kill the task
1251: pthread_mutex_lock(&taskListMutex);
1252: int TaskIndex = runningTaskList.GetIndexOfSdmPid(msgKill.PID);
1253: if (TaskIndex == -1)
1254: {
1255:     printf("Could not find the task to kill \n");
1256:     pthread_mutex_unlock(&taskListMutex);
1257:     return;
1258: }
1259:
1260: runningTaskList[TaskIndex].Kill();
1261:
1262: if(msgKill.killLevel == 1)
1263: {
1264:     debug_f(1, "Permanantly Killing Task with pid: %ld \n", msgKill.PID);
1265:     SDMTaskFinished msgFinished;
1266:     strncpy(msgFinished.filename, runningTaskList[TaskIndex].GetName(),
MAX_FILENAME_SIZE);
1267:     msgFinished.result = 0;
1268:     msgFinished.pid = runningTaskList[TaskIndex].GetSdmPid();
1269:     msgFinished.source.setAddress(Address_PM);
1270:     msgFinished.source.setPort(PORT_PM);
1271:     msgFinished.Send(); //Send the return code to the TM
1272:     MessageSent(&msgFinished);
1273:
1274:     SDMCancelxTEDS msgCancel;
1275:     msgCancel.source.setPort(PORT_PM);
1276:     msgCancel.source.setAddress(Address_PM);
1277:     msgCancel.source.setSensorID(runningTaskList[TaskIndex].GetSdmPid());
1278:     msgCancel.fullCancel = 1;
1279:     msgCancel.Send();
1280:
1281:     runningTaskList.Remove(TaskIndex); //This should keep a task from being reposted

```

```

1282: }
1283:
1284: pthread_mutex_unlock(&taskListMutex);
1285: }
1286: /* CommandHandler handles the reception of SDMCommand messages and performs the
requested command, if it is applicable.
1287: INPUTS:
1288:     buf - The buffer containing the SDMCommand message.
1289: RETURNS:
1290:     None.
1291: */
1292: void CommandHandler(char *buf)
1293: {
1294:     SDMCommand msgCommand;
1295:     if (msgCommand.Unmarshal(buf) < 0)
1296:     {
1297:         printf("Invalid command message. \n");
1298:         return;
1299:     }
1300:     MessageReceived(&msgCommand);
1301:
1302:     // Enable message logging command
1303:     if (msgCommand.command_id == CMD_ENABLE_LOGGING)
1304:     {
1305:         if (log_service.NeedsInit())
1306:             log_service.SetLogFile("Process Manager Message Log \n", "pmmessages.log");
1307:         log_service.AddMessageType(&msgCommand);
1308:     }
1309:     // Disable message logging command
1310:     else if (msgCommand.command_id == CMD_DISABLE_LOGGING)
1311:     {
1312:         log_service.RemoveMessageType(&msgCommand);
1313:     }
1314: }
1315: /*
1316: SubreqstHandler handles the reception of SDMSubreqst messages and adds the subscriber to the
subscription list.
1317: INPUTS:
1318:     buf - The buffer containing the SDMSubreqst message.
1319: RETURNS:
1320:     None.

```



```

1321: */
1322: void SubreqstHandler(char *buf)
1323: {
1324:     SDMSubreqst msgRequest;
1325:     if (msgRequest.Unmarshal(buf) < 0)
1326:     {
1327:         printf("Invalid SDMSubreqst message. \n");
1328:         return;
1329:     }
1330:     MessageReceived(&msgRequest);
1331:     debug_f(1,"Subscription request. \n");
1332:
1333:     // Add the subscription to the PM's sub list
1334:     pthread_mutex_lock(&subscriptionsMutex);
1335:     subscriptions.AddSubscription(msgRequest);
1336:     pthread_mutex_unlock(&subscriptionsMutex);
1337:
1338:     // If request is for performance counters, start system timer
1339:     if (msgRequest.msg_id == NOTI_PERF_COUNTERS)
1340:     {
1341: #ifndef __VXWORKS__
1342:         itimerval interval;
1343:         getitimer(ITIMER_REAL, &interval);
1344:         if (interval.it_value.tv_sec == 0 && interval.it_interval.tv_sec == 0)
1345:         {
1346:             //Time interval for the performance counter publish
1347:             timeval pubInterval;
1348:             pubInterval.tv_sec = 1;
1349:             pubInterval.tv_usec = 0;
1350:
1351:             itimerval timerInterval;
1352:             timerInterval.it_interval = pubInterval;
1353:             timerInterval.it_value = pubInterval;
1354:
1355:             //Set the performance counter timer
1356:             setitimer (ITIMER_REAL, &timerInterval, NULL);
1357:         }
1358:
1359: #else //VxWorks has a different timer API
1360:         itimerspec interval;
1361:         timer_gettime(CLOCK_REALTIME, &interval);

```

```

1362:     if (interval.it_value.tv_sec == 0 && interval.it_interval.tv_sec == 0)
1363:     {
1364:         //Time interval for the publish interval of the performance counter
1365:         itimerspec timerInterval;
1366:         timerInterval.it_value.tv_sec = 1;
1367:         timerInterval.it_value.tv_nsec = 0;
1368:
1369:         //Set the performance counter timer
1370:         timer_settime(CLOCK_REALTIME, 0, &timerInterval, NULL);
1371:     }
1372: #endif
1373: }
1374:
1375: }
1376: /*
1377:  DetelesubHandler deletes a subscription from the ProcessManager's subscription list.
1378:  INPUT:
1379:      buf - The buffer containing the SDMDeletesub message.
1380:  RETURNS:
1381:      None.
1382: */
1383: void DeletesubHandler(char *buf)
1384: {
1385:     SDMDeletesub msgDelete;
1386:     if (msgDelete.Unmarshal(buf) < 0)
1387:     {
1388:         printf("Invalid SDMDeletesub message. \n");
1389:         return;
1390:     }
1391:     MessageReceived(&msgDelete);
1392:     debug_f(1,"Delete subscription request. \n");
1393:
1394:     // Remove the subscription
1395:     pthread_mutex_lock(&subscriptionsMutex);
1396:     subscriptions.RemoveSubscription(msgDelete);
1397:     pthread_mutex_unlock(&subscriptionsMutex);
1398: }
1399: /*
1400:  HeartbeatHandler handles the reception of SDMHeartbeat messages from the TaskManager and
  responds

```

```

1401: to them with an SDMHeartbeat message. This mechanism is used to detect ProcessManager
failure.
1402: INPUTS:
1403:     buf - The buffer containing the SDMHeartbeat message
1404: RETURNS:
1405:     void
1406: */
1407: void HeartbeatHandler (char * buf)
1408: {
1409:     SDMHeartbeat msgHeartBeat;
1410:     if (msgHeartBeat.Unmarshal(buf) < 0)
1411:     {
1412:         printf("Invalid SDMHeartbeat message. \n");
1413:         return;
1414:     }
1415:     MessageReceived(&msgHeartBeat);
1416:
1417:     if (msgHeartBeat.source.getPort() == PORT_TM)
1418:     {
1419:         debug_f(3,"Heartbeat message received (TM). \n");
1420:         // Reply back to the TM
1421:         msgHeartBeat.source.setAddress(Address_PM);
1422:         msgHeartBeat.source.setPort(PORT_PM);
1423:         msgHeartBeat.SendTo(TaskManager);
1424:         MessageSent(&msgHeartBeat);
1425:     }
1426:     else if (msgHeartBeat.source.getPort() == PORT_PM_MONITOR)
1427:     {
1428:         debug_f(3,"Heartbeat message received (PM monitor). \n");
1429:         SDMComponent_ID monitorId (msgHeartBeat.source);
1430:         // Reply back to the monitor process
1431:         msgHeartBeat.source.setAddress(Address_PM);
1432:         msgHeartBeat.source.setPort(PORT_PM);
1433:         msgHeartBeat.SendTo(monitorId);
1434:         MessageSent(&msgHeartBeat);
1435:     }
1436:     else
1437:     {
1438:         debug_f(3,"Heartbeat message received (PID %lu). \n", msgHeartBeat.source.getSensorID());
1439:         // Make note of the heartbeat
1440:         pthread_mutex_lock(&taskListMutex);

```

```

1441:    runningTaskList.HeartbeatReceived(msgHeartBeat.source.getSensorID());
1442:    pthread_mutex_unlock(&taskListMutex);
1443: }
1444: }
1445:
1446: void DMLeaderHandler(char* buf)
1447: {
1448:     SDMDMLLeader temp;
1449:     temp.Unmarshal(buf);
1450:     if(temp.running_flag == 't')
1451:     {
1452:         TaskManager = temp.source;
1453:         printf("New TM Address Received: 0x%lx \n", TaskManager.getAddress());
1454:     }
1455: }
1456:
1457: /*
1458:  SendRunningTasks sends to the TaskManager, all running tasks on this PM node. This routine is
called when the TM
1459:  has experienced a fault and has died. Each PM will then call this routine which allows the TM to
rebuild its
1460:  list of running tasks running in the SDM.
1461:  INPUTS:
1462:      None.
1463:  RETURNS:
1464:      void
1465:  */
1466: void SendRunningTasks(void)
1467: {
1468:     SDMTask msgTask;
1469:
1470:     // The source of this message is this ProcessManager
1471:     msgTask.source.setAddress(Address_PM);
1472:     msgTask.source.setPort(PORT_PM);
1473:
1474:     // Send to the TaskManager, all running tasks on this PM node
1475:     pthread_mutex_lock(&taskListMutex);
1476:     for (unsigned int i = 0; i < runningTaskList.Size(); i++)
1477:     {
1478:         strncpy(msgTask.filename, runningTaskList[i].GetName(), MAX_FILENAME_SIZE);
1479:         msgTask.filename[MAX_FILENAME_SIZE-1] = '\0';

```

```

1480:     msgTask.pid = runningTaskList[i].GetSdmPid();
1481:     msgTask.Send(TaskManager);
1482:     MessageSent(&msgTask);
1483: }
1484: pthread_mutex_unlock(&taskListMutex);
1485: }
1486:
1487: void MessageReceived(SDMmessage* msg)
1488: {
1489:     if (!log_service.IsEmpty())
1490:         log_service.MessageReceived(msg);
1491: }
1492:
1493: void MessageSent(SDMmessage *msg)
1494: {
1495:     // Increment performance counters
1496:     pthread_mutex_lock(&perfCounterMutex);
1497:     total_sent++;
1498:     prevsec_sent++;
1499:     pthread_mutex_unlock(&perfCounterMutex);
1500:
1501:     // Log the message, if the logger is enabled
1502:     if (!log_service.IsEmpty())
1503:         log_service.MessageSent(msg);
1504: }
1505:
1506: /*
1507:  * Determine if a file exists in the current running directory. Used to determine whether a task
1508:  * needs to be
1509:  * added to the pending task list if the IDS is not available.
1510:  */
1510: bool IsFileAvailable(const char* Filename)
1511: {
1512:     if (Filename == NULL) return false;
1513: #ifndef __VXWORKS__
1514:     struct stat file_stats;
1515:
1516:     if (stat(Filename, &file_stats) < 0)
1517:     {
1518:         if (errno == ENOENT)
1519:         {

```

```

1520:         return false;
1521:     }
1522: }
1523: if (file_stats.st_size > 0)
1524:     return true;
1525: return false;
1526: #else
1527: char fullFilename[MAX_FILENAME_SIZE];
1528: strcpy(fullFilename, "/ram0/");
1529: strcat(fullFilename, Filename);
1530: strcat(fullFilename, ".vxe");
1531: if(access(fullFilename, F_OK) == 0)
1532: {
1533:     return true;
1534: }
1535: else
1536: {
1537:     return false;
1538: }
1539: #endif
1540: }
1541:
1542: void RemoveCodeReceiver(const char* strFilename)
1543: {
1544:     pthread_mutex_lock(&codeReceiversMutex);
1545:     const size_t RECEIVER_LIST_SIZE = codeReceivers.size();
1546:     for (unsigned int uiIndex = 0; uiIndex < RECEIVER_LIST_SIZE; uiIndex++)
1547:     {
1548:         if (strcmp (codeReceivers[uiIndex].filename, strFilename) == 0)
1549:         {
1550:             debug_f(3, "Removing code receiving entry for task %s. \n", strFilename);
1551:             close(codeReceivers[uiIndex].fd_code);
1552:             codeReceivers.erase(codeReceivers.begin() + uiIndex, codeReceivers.begin() + uiIndex +
1553: 1);
1553:             break;
1554:             //fall through
1555:         }
1556:     }
1557:     pthread_mutex_unlock(&codeReceiversMutex);
1558: }
1559:

```

File: sdm/pm/pm.h

```
1: #ifndef _PM_H_
2: #define _PM_H_
3:
4: #define TM_Mode_ID    1      //ID number used in RegInfo and ReqReg messages for TM's status
    message
5:
6: extern void MessageSent(SDMmessage*);
7: extern void MessageReceived(SDMmessage*);
8: extern void SendRunningTasks(void);
9: extern void RegisterPM(void);
10: extern void DoReset(void);
11:
12:
13: void CancelReqRegSubs();
14: void ReqRegSubs();
15: void RegInfoHandler(const char* buf);
16: void DataHandler (const char *buf);
17: void HandleCodeMessage(const char* buf);
18: bool GetCode (const char* filename, long pid, int version, int fd_readcode);
19:
20: #endif
```

File: sdm/pm/pm_ids.h

```
1: #ifndef _PM_IDS_H_
2: #define _PM_IDS_H_
3:
4: #define TM_Mode_ID    1      //ID number used in RegInfo and ReqReg messages for TM's status
    message
5:
6: void HandleCodeMessage(const char* buf);
7: void ReqRegSubs(void);
8: void TaskHandler(const char* buf);
9: void RegInfoHandler(const char* buf);
10: void DataHandler (const char *buf);
11: void CancelReqRegSubs();
12: bool GetCode (const char* filename, long pid, int version, int fd_readcode);
13:
14: #endif
```


Listing from directory: sdm/sm

File: sdm/sm/sensor.cpp

```
1: #include "sensor.h"
2: #include "../common/message_defs.h"
3:
4: #include <string.h>
5: #include <stdio.h>
6: #include <unistd.h>
7: #include <errno.h>
8: #include <ctype.h>
9: #include <stdlib.h>
10: #include <sys/time.h>
11: #include "../common/message/SDMxTEDS.h"
12: #include "../common/message/SDMCancelxTEDS.h"
13: #include "../common/message/SDMTat.h"
14: #include "../common/message/SDMHello.h"
15: #include "SensorMonitor.h"
16:
17: #define MAX_TRIES 30
18:
19: Sensor::Sensor():
20:     device(ASIM()),
21:     sensor_id(),
22:     device_xTEDS(NULL),
23:     subscriptions(SubscriptionManager()),
24:     sensor_mutex(),
25:     connected(false),
26:     m_Ready(false),
27:     debug_level(0),
28:     m_USBMonitorThread(),
29:     m_Monitor(NULL)
30: {
31:     device_name[0] = '\0';
32:     pthread_mutex_init(&sensor_mutex, NULL);
33:     sensor_id.setPort(PORT_SM);
34: }
35:
36: Sensor::Sensor(long id):
37:     device(ASIM()),
```

```

38: sensor_id(),
39: device_xTEDS(NULL),
40: subscriptions(SubscriptionManager()),
41: sensor_mutex(),
42: connected(false),
43: m_Ready(false),
44: debug_level(0),
45: m_USBMonitorThread(),
46: m_Monitor(NULL)
47: {
48: device_name[0] = '\0';
49: pthread_mutex_init(&sensor_mutex, NULL);
50: sensor_id.setPort(PORT_SM);
51: Open(id);
52: }
53:
54: // It doesn't really make sense to copy or assign a Sensor object. Therefore
55: // Copy Constructor and operator= functions are declared, but not defined;
56: // creating a linker error if the developer tries to use them.
57:
58: // Sensor::Sensor(const Sensor& a):
59: //   subscriptions(a.subscriptions),
60: //   device(a.device),
61: //   sensor_id(a.sensor_id),
62: //   device_xTEDS(a.device_xTEDS),
63: //   mutex(a.mutex),
64: //   connected(a.connected),
65: //   debug_level(a.debug_level)
66: // {
67: // }
68: //
69: // Sensor& Sensor::operator=(const Sensor& a)
70: // {
71: //   subscriptions = a.subscriptions;
72: //   device = a.device;
73: //   sensor_id = a.sensor_id;
74: //   mutex = a.mutex;    //this is a pointer but the copy should point to the
75: //                       //same mutex
76: //   connected = a.connected;
77: //   return *this;
78: // }

```

```

79:
80: Sensor::~Sensor()
81: {
82:   pthread_mutex_destroy(&sensor_mutex);
83: }
84:
85: void Sensor::SetDebug(int new_debug_level)
86: {
87:   debug_level = new_debug_level;
88:   device.SetDebug(new_debug_level);
89: }
90:
91: /** Open sensor for use
92:
93:  Open sets the sensor's SensorID number and saves the path name of the sensor.
94:  This function is called from the SensorManager.
95:
96:  INPUTS:
97:  @param id - The sensor ID number to be assigned for this sensor.
98:
99:  RETURNS:
100:  @return bool - True on success, false otherwise.
101:  */
102: bool Sensor::Open(long id)
103: {
104:   //set sensor id
105:   sensor_id.setSensorID(id+SM_OFFSET);
106:   //build device_name string
107: #ifdef WIN32
108:   sprintf(device_name, "\\.\ASIM-0%ld",id+1);
109: #else
110:   sprintf(device_name, "/dev/asim%ld",id);
111: #endif
112:   return true;
113: }
114:
115: /** Monitor the USB channel
116:
117:  USBMonitor is a thread function that monitors the USB channel through which
118:  the SensorManager and the sensor device communicate. If a device is not
119:  available, this function tries to open a handle. If a device is connected,

```

```

120: this function attempts to read data from it. If a device is connected and
121: then loses connection, this function will cancel the device from the
122: DataManager.
123:
124: INPUTS:
125: @param arg - The Sensor object representing the sensor connected.
126:
127: RETURNS:
128: @return void * - Always NULL.
129: */
130: void* Sensor::USBMonitor(void* arg)
131: {
132:     USBMonitorThreadArgs* ThreadArgs = (USBMonitorThreadArgs*)arg;
133:     Sensor* sensor = ThreadArgs->SensorInstance;
134:     SensorMonitor* monitor = ThreadArgs->MonitorInstance;
135:     int ID = ThreadArgs->SensorID;
136:     delete ThreadArgs;
137:
138:     unsigned short length;
139:     unsigned char buf[BUFSIZE];
140:     //open ASIM device
141:
142:     // Set the thread's cancel state and type
143:     if ( 0 != pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL) ||
144:         0 != pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL) )
145:     {
146:         printf("Could not set the thread's cancel state/type in USBMonitor. \n");
147:         return NULL;
148:     }
149:
150:     if(sensor->debug_level>=2)
151:         printf("Monitoring %s \n",sensor->device_name);
152:
153:     while(1)
154:     {
155:         fflush(NULL);
156:         pthread_testcancel();
157:         if(sensor->connected)
158:         {
159:             /*Verify that this sensor is still connected*/
160:             sensor->connected = sensor->device.VerifyConnection();

```

```

161:  /*If the sensor is still connected, attempt to read from it*/
162:  if(sensor->connected)
163:  {
164:  if(sensor->debug_level>=3)
165:    printf("starting read on %s \n",sensor->device_name);
166:
167:  sensor->Read(length, buf, sizeof(buf));
168:  }
169:  /*If the device has lost connected, cancel it*/
170:  else
171:  {
172:  if(sensor->debug_level>=0)
173:  {
174:    printf("%s disconnected \n",sensor->device_name);
175:    fflush(NULL);
176:  }
177:  sensor->Cancel();
178:  sensor->m_Ready = false;
179:  sensor->device.Close();
180:  }
181: }
182: /*If the sensor is not connected, attempt to open a handle to it.*/
183: else
184: {
185:  /*If the sensor exists, register it*/
186:  if(sensor->device.Open(sensor->device_name))
187:  {
188:  if(sensor->debug_level>=0)
189:  {
190:    printf("%s connected \n",sensor->device_name);
191:    fflush(NULL);
192:  }
193:  sensor->connected = true;
194:  monitor->SensorRegistering(ID);
195:  if(sensor->Register(ID))
196:  {
197:    printf("%s registered \n",sensor->device_name);
198:    monitor->SensorRegistered(ID);
199:    sensor->m_Ready = true;
200:    fflush(NULL);
201:  }

```

```

202:     else
203:     {
204:         printf("%s failed to register \n",sensor->device_name);
205:         fflush(NULL);
206:         sensor->m_Ready = false;
207:         sensor->device.Close();
208:         sensor->connected = false;
209:     }
210:
211:     }
212:     /*Otherwise, close the handle*/
213:     else
214:     {
215:         if(sensor->debug_level>=6)
216:         {
217:             printf("No device connected at %s \n",sensor->device_name);
218:             fflush(NULL);
219:         }
220:         sensor->device.Close();
221:         sleep(1);
222:     }
223: }
224: }
225: return NULL;
226: }
227:
228: /** Start the USB listener thread.
229:
230: StartUSBMonitor starts the USB listener thread for sensor communications,
231: by spawning off a new thread.
232: INPUTS:
233: @param id - The SensorID number of the sensor device.
234:
235: RETURNS:
236: @return pthread_t* - A pointer to the pthread started.
237: */
238: pthread_t* Sensor::StartUSBMonitor(long id, SensorMonitor* Monitor)
239: {
240:     //static pthread_t USBMonitor_thread;
241:     m_Monitor = Monitor;
242:

```

```

243: int result;
244: Open(id);
245: result = pthread_create(&m_USBMonitorThread,NULL,&USBMonitor,
246:         new USBMonitorThreadArgs(this, Monitor, id));
247: if(result < 0)
248: {
249:     perror("Could not start ASIM monitor");
250: }
251: return &m_USBMonitorThread;
252: }
253:
254: /** Remove a sensor from registration and listening.
255:
256: Kill the listener thread, and removes the subscription for the sensor.
257:
258: INPUTS:
259: @param Disconnected - is the sensor currently disconnected.
260: */
261: void Sensor::CancelSensor(bool Disconnected)
262: {
263:     if (Disconnected)
264:         printf("%s disconnected \n", device_name);
265:     // Kill the listener thread
266:     if (0 != pthread_cancel(m_USBMonitorThread))
267:     {
268:         printf("Could not cancel USBMonitor thread. \n");
269:     }
270:     // Cancel xTEDS and subscriptions
271:     Cancel();
272: }
273:
274: /** Check if sensor still connected
275: *
276: * RETURNS:
277: * @return true if connected, otherwise false
278: *
279: * NOTES:
280: * Alternative to device.VerifyConnection(). Trying device.VerifyConnection()
281: * while another thread is block on a read for the device resets the ASIM.
282: */
283: bool Sensor::CheckConnection()

```

```

284: {
285: // This is sort of an ugly hack... Trying device.VerifyConnection() while
286: // another thread is block on a read for the device resets the ASIM. This
287: // method will check to see if the handle still exists in /dev. This is used
288: // to verify that an ASIM hasn't disconnected and if it has, the calling
289: // thread can cancel it. The device driver doesn't cancel
290: // an ASIM if it is connected, but the handle disappears in /dev.
291:
292: char Command[64];
293: snprintf(Command, sizeof(Command), "ls %s > /dev/null", device_name);
294: if (0 != system(Command))
295:     return false;
296:
297: return true;
298: }
299:
300: /** TimeAtTone sends a TAT message down to the attached ASIM.
301: *
302: * INPUTS:
303: * @param seconds - The seconds value for the time.
304: * @param useconds - The microseconds value for the time.
305: *
306: * RETURNS:
307: * @return void
308: */
309: void Sensor::TimeAtTone(unsigned long seconds,unsigned long useconds)
310: {
311: if(!m_Ready)
312:     return;
313:
314: if(debug_level>=1)
315:     printf("%s : Time At Tone %lu sec %lu usec \n",device_name,seconds,useconds);
316:
317: pthread_mutex_lock(&sensor_mutex);
318: if (!device.TimeAtTone(seconds,useconds))
319:     printf("%s : Error sending time at tone message. \n", device_name);
320: pthread_mutex_unlock(&sensor_mutex);
321: }
322:
323: /** Read data from connected ASIM.
324: *

```



```

325: * Read attempts to read data from the connected ASIM. If the message is data,
326: * it is published to all interested subscribers, otherwise
327: * the message received is printed to the screen (if debug is high enough).
328: *
329: * INPUTS:
330: * @param buf [output] - The buffer in which to store the message.
331: * @param length [output] - The number of bytes stored in buf.
332: *
333: * RETURNS:
334: * @return char - The type of ASIM message received.
335: */
336: char Sensor::Read(unsigned short& length, unsigned char* buf, int bufsize)
337: {
338:   char msg_type;
339:   int error_code;
340:   char error_str[80];
341:   int i;
342:
343:   if(!connected)
344:     return ASIM_ERROR;
345:
346:   memset(buf, 0, bufsize); //Clear out the message buffer
347:   msg_type = device.Read(length,buf,bufsize);
348:   error_code = errno;
349:
350:   switch(msg_type)
351:   {
352:   case ASIM_STATUS:
353:     if (buf[0]&0x80) //first bit is set for an error
354:     {
355:       if(debug_level>=0)
356:       {
357:         printf("ASIM %s status ERROR: %hhd \n",device_name,buf[0]);
358:       }
359:       if(debug_level>=2)
360:       {
361:         if(buf[0]&0x40) //illegal command bit
362:         {
363:           printf("\tIllegal or unrecognized command \n");
364:         }
365:         if(buf[0]&0x20) //self test failure bit

```

```

366:  {
367:    printf(" \tSelf Test failed \n");
368:  }
369:  }
370: }
371: else
372: {
373:   if(debug_level>=1)
374:   {
375:     printf("ASIM %s status OK: %hhhd \n",device_name,buf[0]);
376:   }
377: }
378: if(debug_level>=2)
379: {
380:   if(buf[0]&0x10) //mode bit
381:   {
382:     printf(" \tASIM mode: operational \n");
383:   }
384:   else
385:   {
386:     printf(" \tASIM mode: idle \n");
387:   }
388: }
389: break;
390:
391: case ASIM_XTEDS:
392:   if(debug_level>=1)
393:   {
394:     printf("%s xTEDS (%d) \n",device_name,length);
395:     fflush(NULL);
396:     if(debug_level>=2)
397:     {
398:       for(i=0;i<length;++i)
399:       {
400:         if(isprint(buf[i])||isspace(buf[i]))
401:         {
402:           printf("%c",buf[i]);
403:         }
404:         else
405:         {
406:           printf(" \nERROR %s xTEDS contains invalid character 0x%x \n",

```

```

407:         device_name,buf[i]);
408:     }
409: }
410: }
411: }
412: break;
413:
414: case ASIM_XTEDS_ID_PAIR:
415:     if(debug_level>=1)
416:     {
417:         printf("%s xTEDS_PID (%d) \n",device_name,length);
418:         fflush(NULL);
419:         if(debug_level>=2)
420:         {
421:             printf(" (ID=0x");
422:             for(i = 0; i < (length > 4 ? 4 : length);++i)
423:                 printf("%x",buf[i]);
424:             printf(") \n");
425:             for( ;i<length;++i)
426:             {
427:                 if(isprint(buf[i])||isspace(buf[i]))
428:                 {
429:                     printf("%c",buf[i]);
430:                 }
431:                 else
432:                 {
433:                     printf(" \nWARNING: %s xTEDS contains invalid character 0x%x \n",
434:                         device_name,buf[i]);
435:                 }
436:             }
437:         }
438:     }
439:     break;
440:
441: case ASIM_DATA:
442:     if(debug_level>=2)
443:     {
444:         printf("%s produced message %hhhd \n",device_name,buf[0]);
445:     }
446:     pthread_mutex_lock(&sensor_mutex);
447:     if(!subscriptions.Publish(buf[0],buf[1],(char*)(buf+2),length-2))

```

```

448: {
449:     if(debug_level>=1)
450:     {
451:         printf(" \t%s:%hhhd had no subscribers \n",device_name,buf[0]);
452:         printf(" \tCanceling %s:%hhhd \n",device_name,buf[0]);
453:     }
454:     device.Cancel(buf[0], buf[1]);
455: }
456: else
457: {
458:     MessageSent(subscriptions.GetLastPublished());
459: }
460: pthread_mutex_unlock(&sensor_mutex);
461: break;
462:
463: case ASIM_TIME_AT_TONE:
464: {
465:     // A modification for Fronterhouse. The ASI protocol doesn't define an
466:     // ASIM producing a Time-At-Tone, however, the GPS SPA-U ASIM will. This
467:     // will send it to the SM, which will in turn send it to all ASIMs connected
468:     // to the SM.
469:
470:     SDMTat TATMsg;
471:     char SendBuffer[BUFSIZE];
472:
473:     TATMsg.destination.setSensorID(0);// Send to all ASIMs
474:     TATMsg.seconds = GET_UINT(buf);
475:     TATMsg.useconds = GET_UINT(buf+4);
476:
477:     int TATLength = TATMsg.Marshal(SendBuffer);
478:     if (TATLength < 0)
479:     {
480:         printf("Error marshalling TAT message. \n");
481:         return ASIM_ERROR;
482:     }
483:
484:     // Send to the SensorManager listen thread
485:     if (UDPsendto("127.0.0.1", PORT_SM, SendBuffer, TATLength) < 0)
486:     {
487:         printf("Error sending TAT message. \n");
488:         return ASIM_ERROR;

```

```

489:  }
490:  }
491:  break;
492:
493:  case ASIM_VERSION:
494:    if(debug_level>=1)
495:      printf("%s running ASIM version %hhd \n",device_name,buf[0]);
496:    break;
497:
498:  case ASIM_ERROR:
499:    if(debug_level>=0) //display errors
500:    {
501:      printf("%s read error: %s \n",device_name,strerror_r(error_code,error_str,80));
502:    }
503:    break;
504:
505:  case ASIM_TIMEOUT:
506:    if(debug_level>=3) //only display timeout errors on debug 3
507:    {
508:      printf("%s read timeout: %s \n",
509:        device_name,strerror_r(error_code,error_str,80));
510:    }
511:    break;
512:
513: #ifdef WIN32
514:  case NULL: /*Default return for no message available in Win32*/
515:    usleep(10000); /*Give time for a message to be produced*/
516:    break;
517: #endif
518:  default:
519:    if(debug_level>=0)
520:      printf("%s produced an undefined message 0x%x \n",device_name,msg_type);
521:    break;
522:  }
523:  return msg_type;
524: }
525:
526: /** Register initializes the sensor, requests its xTEDS, and registers the
527:  * sensor with the SDM.
528:  *
529:  * INPUTS:

```

```

530: * None.
531: * RETURNS:
532: * @return bool - True on success, false on failure.
533: */
534: bool Sensor::Register(int sensorIndex)
535: {
536:     SDMxTEDS xTEDS;
537:     unsigned short length;
538:     char buf[BUFSIZE];
539:     bool isValid = false;
540:     char* end_xTEDS;    //pointer to the </xteds> tag
541:     char status;
542:     int register_tries;
543:     int sleep_time;
544:
545:     //
546:     // Initialize
547:     //
548:     if(debug_level>=2)
549:         printf("Initializing %s \n",device_name);
550:
551:     device.Initialize();
552:     status = Read(length,(unsigned char*)buf, sizeof(buf));
553:     if(status == ASIM_ERROR)
554:         return false;
555:
556:     //
557:     // Version Request
558:     //
559:     if(debug_level>=2)
560:         printf("Requesting ASIM version for %s \n",device_name);
561:     device.ReqVersion();
562:     status = Read(length,(unsigned char*)buf, sizeof(buf));
563:     if(status == ASIM_ERROR)
564:         return false;
565:
566:     //
567:     // xTEDS Request
568:     //
569:     if(debug_level>=2)
570:         printf("Requesting xTEDS for %s \n",device_name);

```

```

571: device.ReqxTEDS();
572: register_tries = 1;
573: sleep_time = 0;
574:
575: while(!isValid)
576: {
577:     status = Read(length,(unsigned char*)buf, sizeof(buf));
578:     if(status==ASIM_ERROR)
579:         return false;
580:
581:     if(status!=ASIM_XTEDS && status!=ASIM_XTEDS_ID_PAIR)
582:     {
583:         //Occasionally on the first attempt, a version request may have just
584:         // been read, so don't output the below error message if not really an
585:         // error on the first attempt
586:         if(debug_level>=0 && register_tries != 1)
587:         {
588:             printf("ERROR: reading xTEDS for %s failed, retrying - status is %hhx \n",
589:                 device_name,status);
590:         }
591:
592:         if(register_tries > MAX_TRIES)
593:         {
594:             printf("ERROR: bad device %s \n",device_name);
595:             return false;
596:         }
597:         sleep_time += register_tries;    //increase the wait between tries
598:         register_tries++;
599:         sleep(sleep_time);
600:         if (debug_level >= 2)
601:         {
602:             printf("Re-requesting xTEDS for %s \n",device_name);
603:         }
604:         device.ReqxTEDS();           //re-request xTEDS
605:     }
606:     else
607:     {
608:         //look for </xTEDS> to insure that the entire xTEDS was read
609:         end_xTEDS = NULL;
610:         end_xTEDS = strstr(buf+4,"</xTEDS>"); //Add four because if a PID is
611:                                                // present, it could contain 0x00

```

```

612:                                     // (null-term)
613:  if(end_xTEDS!=NULL)
614:  isValid = true;
615:  else
616:  {
617:  if(debug_level>=0)
618:  {
619:    printf("ERROR: read invalid xTEDS for %s \n",device_name);
620:    if(debug_level>=2)
621:    {
622:      printf("Requesting xTEDS for %s \n",device_name);
623:    }
624:  }
625:
626:  if(register_tries > MAX_TRIES)
627:  {
628:    printf("ERROR: bad device %s \n",device_name);
629:    return false;
630:  }
631:
632:  sleep_time += register_tries;    //increase the wait between tries
633:  register_tries++;
634:  sleep(sleep_time);
635:  device.ReqxTEDS();              //re-request xTEDS
636:  }
637: }
638: }
639:
640: //Free the old xTEDS, if this is a re-reg
641: if (device_xTEDS != NULL)
642: {
643:  delete [] device_xTEDS;
644:  device_xTEDS = NULL;
645: }
646:
647:
648: //TODO: What is this whole ID pair thing?
649: //If this was an xTEDS reply with a corresponding unique ID
650: if (status == ASIM_XTEDS_ID_PAIR)
651: {
652:  device_xTEDS = new char[length+1-4];    //Allocate xTEDS buffer, without

```



```

653:                                     // ID number
654:  strncpy(device_xTEDS, buf+4, length-4);    //Copy the xTEDS, without the ID
655:                                     // number
656:  device_xTEDS[length] = '\0';              //End string
657:  //xTEDS.pid = GET_INT(buf);                //Copy out the PID identifier of
658:                                     // this ASIM
659:  //memcpy(xTEDS.xTEDS, buf+4, length - 4);   //Copy the xTEDS into the
660:                                     //SDMxTEDS message, without the ID
661: }
662: //This is just a plain old xTEDS reply
663: else
664: {
665:  device_xTEDS = new char[length+1];        //Allocate the xTEDS buffer
666:  strncpy(device_xTEDS,buf, length);          //Copy the xTEDs
667:  device_xTEDS[length] = '\0';              //End string
668: }
669:
670: SDMHHello helloMsg;
671: helloMsg.type = 'D';
672: helloMsg.source.setPort(PORT_SM);
673: helloMsg.source.setSensorID(sensorIndex);
674: helloMsg.Send();
675: unsigned int sendTime = 0;
676: if(m_Monitor != NULL)
677: {
678:  sendTime = m_Monitor->SetHelloSendTime(sensorIndex);
679: }
680:
681:
682: if(debug_level >= 0)
683: {
684:  printf("SDMHHello msg sent for sensor at index: %i at time: %i \n", sensorIndex, sendTime);
685: }
686:
687: if(debug_level>=2)
688: {
689:  printf("USB path of %s:%s \n",device_name,xTEDS.SPA_node);
690: }
691:
692: return true;
693: }

```

```

694:
695: /** ReRegister the ASIM.
696: * ReRegister re-registers an ASIM with the SDM by resending its xTEDS. A
697: * device is reregistered when the DataManager fails and a new DM has been
698: * elected.
699: *
700: * INPUTS:
701: * None.
702: * RETURNS:
703: * @return void
704: */
705: void Sensor::ReRegister()
706: {
707: //Reregister the xTEDS of the current device
708: SDMxTEDS xteds_msg;
709: if (m_Ready)
710: {
711: strcpy(xteds_msg.SPA_node, device.USBLocation());
712: xteds_msg.source = sensor_id;
713: strcpy(xteds_msg.xTEDS, device_xTEDS);
714: xteds_msg.Send();
715: MessageSent(&xteds_msg);
716: }
717: //Cancel all subscriptions, which are no longer applicable
718: pthread_mutex_lock(&sensor_mutex);
719: subscriptions.ClearAllSubscriptions();
720: pthread_mutex_unlock(&sensor_mutex);
721: }
722:
723: /** Service sends a service request message down to an ASIM.
724: *
725: * INPUTS:
726: * @param request - The SDMSerreqst message whose data will be sent.
727: * RETURNS:
728: * @return void
729: */
730: void Sensor::Service(SDMSerreqst& request)
731: {
732: if(!m_Ready)
733: return;
734:

```

```

735: if(debug_level>=1)
736:     printf("%s recieved service request. \n",device_name);
737:
738: pthread_mutex_lock(&sensor_mutex);
739: subscriptions.AddSubscription(request);
740: device.Command(request.command_id.getInterface(),
741:     request.command_id.getMessage(),
742:     request.length,
743:     (unsigned char*)request.data);
744: pthread_mutex_unlock(&sensor_mutex);
745: }
746:
747: /** Command sends a command message down to an ASIM.
748:  *
749:  * INPUTS:
750:  * @param command - The SDMCommand message whose data will be sent.
751:  * RETURNS:
752:  * @return void
753:  */
754: void Sensor::Command(SDMCommand& command)
755: {
756: if(!m_Ready)
757:     return;
758:
759: if(debug_level>=1)
760:     printf("%s recieved command for interface id: %d command id: %d \n",
761:         device_name,
762:         command.command_id.getInterface(),
763:         command.command_id.getMessage());
764:
765: pthread_mutex_lock(&sensor_mutex);
766: device.Command(command.command_id.getInterface(),
767:     command.command_id.getMessage(),
768:     command.length,(unsigned char*)command.data);
769: pthread_mutex_unlock(&sensor_mutex);
770: }
771:
772: /** Cancel an ASIM from the SDM system.
773:  *
774:  * Cancel cancels an ASIM from the SDM system by sending an SDMCancelxTEDS
775:  * message to the DataManager.

```

```

776: *
777: * INPUTS:
778: * None.
779: * RETURNS:
780: * @return void
781: */
782: void Sensor::Cancel()
783: {
784:   if(debug_level>=2)
785:     printf("Canceling xTEDS for %s \n",device_name);
786:
787:   SDMCancelxTEDS cancel;
788:   cancel.source = sensor_id;
789:   cancel.Send();
790:   MessageSent(&cancel);
791:
792:   pthread_mutex_lock(&sensor_mutex);
793:   connected = false;
794:   subscriptions.ClearAllSubscriptions();
795:   pthread_mutex_unlock(&sensor_mutex);
796: }
797:
798: /** Subscribe sends a subscription request (for a data stream) down to the ASIM.
799: *
800: * INPUTS:
801: * @param request - The SDMSubreqst message whose data is sent.
802: *
803: * RETURNS:
804: * @return void
805: */
806: void Sensor::Subscribe(SDMSubreqst& request)
807: {
808:   if(!m_Ready)
809:     return;
810:
811:   if(debug_level>=1)
812:     printf("%s recieved subscription request. \n",device_name);
813:
814:   pthread_mutex_lock(&sensor_mutex);
815:   subscriptions.AddSubscription(request);
816:   device.ReqData(request.msg_id.getInterface(),

```

```

817:         request.msg_id.getMessage(),
818:         request.destination.getAddress(),
819:         request.destination.getPort());
820: pthread_mutex_unlock(&sensor_mutex);
821: }
822:
823: /** CancelSubscription sends a cancel subscription request down to the ASIM.
824:  *
825:  * INPUTS:
826:  * @param request - The SDMDeltesub message whose data will be sent.
827:  *
828:  * RETURNS:
829:  * @return void
830:  */
831: void Sensor::CancelSubscription(SDMDeltesub& request)
832: {
833:     if(!m_Ready)
834:         return;
835:
836:     if(debug_level>=1)
837:         printf("%s recieved cancel subscription request. \n",device_name);
838:
839:     pthread_mutex_lock(&sensor_mutex);
840:     subscriptions.RemoveSubscription(request);
841:     pthread_mutex_unlock(&sensor_mutex);
842:
843: }

```

File: sdm/sm/sm.h

```
1: #ifndef _SDM_SM_H_
2: #define _SDM_SM_H_
3:
4: #include "sensor.h"
5: #include "../common/message/SDMmessage.h"
6: #include "../common/message/SDMReady.h"
7: #include "../common/message/SDMSubreqst.h"
8: #include "../common/message/SDMDeletesub.h"
9: #include "../common/message/SDMSerreqst.h"
10: #include "../common/message/SDMCommand.h"
11: #include "../common/message/SDMTat.h"
12: #include "../common/message/SDMError.h"
13: #include "../common/message/SDMCancelxTEDS.h"
14: #include "../common/message/SDMRegInfo.h"
15: #include "../common/message/SDMReqReg.h"
16: #include "../common/message/SDMConsume.h"
17: #include "../common/message/SDMCancel.h"
18: #include "../common/message/SDMxTEDS.h"
19: #include "../common/message/SDMAck.h"
20: #include "../common/message/SDMHello.h"
21: #include "../common/message/SDMRegister.h"
22: #include "../common/message/SDMID.h"
23: #include "../common/SubscriptionManager/SubscriptionManager.h"
24: #include "../common/MessageLogger/MessageLogger.h"
25: #include "../common/TCPcom.h"
26: #include "../common/Debug.h"
27: #include "../common/version.h"
28: #include "../common/Time/SDMTime.h"
29: #include <pthread.h>
30: #include <unistd.h>
31: #include <string.h>
32: #include <stdio.h>
33: #include <stdlib.h>
34: #include <getopt.h>
35: #include <sys/types.h>
36: #include <sys/socket.h>
37: #include <sys/poll.h>
38: #include <sys/time.h>
39: #include <netinet/in.h>
```

```

40: #include <arpa/inet.h>
41: #include <signal.h>
42: #ifndef WIN32
43: #include <net/if.h>
44: #endif
45:
46: #define TM_Mode_ID 1
47:
48: //SM message ids
49: const unsigned long SENSOR_MANAGER_SID = 1;
50: const SDMMMessage_ID CMD_ENABLE_LOGGING(3,16);
51: const SDMMMessage_ID CMD_DISABLE_LOGGING(3,17);
52: const SDMMMessage_ID SUB_PERFORMANCE_COUNTERS(2,13);
53:
54:
55: // Function prototypes
56: void VerifyDM(void);
57: void* UDPMonitor(void*);
58: void* TCPMonitor(void*);
59: double GetCurTime();
60: #ifdef WIN32
61: void SigIntHandler(int signum);
62: #else
63: void* SigHandler(void*);
64: #endif
65: void PublishPerformanceCounterMessage(void);
66: void MessageSent(SDMmessage *msg);
67: void MessageReceived(SDMmessage *msg);
68: int GetNodeAddress();
69:
70: #endif
71:

```

File: sdm/sm/SensorRecord.h

```
1: #ifndef _SENSOR_RECORD_H_
2: #define _SENSOR_RECORD_H_
3:
4: const unsigned int SECONDS_INIT_VAL = 0;
5: const int SENSOR_ACK_TIMEOUT = 10;
6: const int SENSOR_ID_TIMEOUT = 15;
7:
8: class SensorRecord
9: {
10: public:
11:   SensorRecord();
12:   ~SensorRecord();
13:
14:   void Registered(bool NewRegistered) { m_Registered = NewRegistered; }
15:   bool Registered() { return m_Registered; }
16:
17:   void SetStartTime(unsigned int Seconds) { m_SecondsStart = Seconds; }
18:   unsigned int StartTime() { return m_SecondsStart; }
19:
20:   void SetHelloTime(unsigned int seconds) { helloSentTime = seconds; }
21:   unsigned int GetHelloTime() { return helloSentTime; }
22:   void SetxTEDSTime(unsigned int seconds) { xTEDSSentTime = seconds; }
23:
24:   void Acked(bool acked) { ackReceived = acked; }
25:   bool Acked() { return ackReceived; }
26:   void RegConfirmed(bool regConf) { regConfirmed = regConf; }
27:   bool RegConfirmed() { return regConfirmed; }
28:
29:   bool HasExpired(unsigned int CurSeconds);
30:
31:   bool CheckAckTimeout(unsigned int curSeconds);
32:   bool CheckIDTimeout(unsigned int curSeconds);
33:
34:
35: private:
36:   bool m_Registered;
37:   unsigned int m_SecondsStart;
38:   unsigned int helloSentTime;
39:   unsigned int xTEDSSentTime;
```



```
40:  bool ackReceived;  
41:  bool regConfirmed;  
42: };  
43:  
44: #endif  
45:
```

File: sdm/sm/SensorMonitor.cpp

```
1: #include <sys/time.h>
2: #include <unistd.h>
3: #include <string.h>
4: #include "SensorMonitor.h"
5: #include "../common/message/SDMHello.h"
6: #include "../common/message/SDMxTEDS.h"
7: #ifdef WIN32
8: # include <winsock2.h>
9: #endif
10: // #define DEBUG_MONITOR 1
11:
12: const int TIMEOUT_INTERVAL = 6; // In seconds
13: const int SENSOR_CHECK_INTERVAL = 5;
14:
15:
16: SensorMonitor::SensorMonitor() : m_SensorMonitorThread(), m_NumSensors(0),
m_Sensors(NULL), m_SensorDataMutex(), m_SensorData(NULL)
17: {
18: pthread_mutex_init(&m_SensorDataMutex, NULL);
19: }
20:
21: SensorMonitor::~SensorMonitor()
22: {
23: delete [] m_Sensors;
24: delete [] m_SensorData;
25:
26: pthread_mutex_destroy(&m_SensorDataMutex);
27: }
28:
29: bool SensorMonitor::InitializeSensors(unsigned int NumSensors, bool Prompt, int DebugLevel)
30: {
31: m_NumSensors = NumSensors;
32: m_Sensors = new Sensor[NumSensors];
33: m_SensorData = new SensorRecord[NumSensors];
34:
35: for(unsigned int i = 0; i < NumSensors; ++i)
36: {
37:     bool init_sensor = true;
38:     char response;
```

```

39:     if(Prompt)
40:     {
41:         printf("Initialize sensor /dev/asim%d (y/n)? ",i);
42:         scanf("%c",&response);
43:         getchar();
44:         if((response == 'Y')||(response == 'y'))
45:         {
46:             printf(" \tStarting /dev/asim%d \n",i);
47:         }
48:         else
49:         {
50:             init_sensor = false;
51:         }
52:     }
53:     if(init_sensor)
54:     {
55:         m_Sensors[i].StartUSBMonitor(i, this);
56:         m_Sensors[i].SetDebug(DebugLevel);
57:     }
58: }
59:
60: // Start the monitor thread
61: if (0 != pthread_create(&m_SensorMonitorThread, NULL, &SensorMonitorFunc, this))
62: {
63:     printf("Error starting sensor monitor thread. \n");
64:     return false;
65: }
66:
67: // Detach the monitor thread
68: if (0 != pthread_detach(m_SensorMonitorThread))
69: {
70:     printf("Error detaching sensor monitor thread. \n");
71:     return false;
72: }
73: return true;
74: }
75:
76:
77: unsigned int GetCurSeconds()
78: {
79: struct timeval Time;

```

```

80: gettimeofday(&Time, NULL);
81: return Time.tv_sec;
82: }
83:
84: void* SensorMonitor::SensorMonitorFunc(void* SensorMonitorInstance)
85: {
86: SensorMonitor* Monitor = static_cast<SensorMonitor*>(SensorMonitorInstance);
87:
88: while (1)
89: {
90:     unsigned int CurSeconds;
91:     for (int i = 0; i < Monitor->m_NumSensors; i++)
92:     {
93: #ifdef DEBUG_MONITOR
94:         printf("Checking sensor at index: %i \n", i);
95: #endif
96:         //
97:         // First check for registration timeouts
98:         //
99:         bool ThreadReset = false;
100:         pthread_mutex_lock(&Monitor->m_SensorDataMutex);
101:         if (!Monitor->m_SensorData[i].Registered() && Monitor->m_SensorData[i].StartTime()
!= SECONDS_INIT_VAL)
102:         {
103:             // Check for a registration timeout
104:             CurSeconds = GetCurSeconds();
105:             if (CurSeconds > (Monitor->m_SensorData[i].StartTime() +
TIMEOUT_INTERVAL))
106:             {
107: #ifdef DEBUG_MONITOR
108:                 printf("Sensor registration timeout, restarting. \n");
109: #endif
110:                 // Registration has timed out
111:                 ThreadReset = true;
112:
113:                 Monitor->m_Sensors[i].CancelSensor(false);
114:                 Monitor->m_Sensors[i].StartUSBMonitor(i, Monitor);
115:             }
116:         }
117:         pthread_mutex_unlock(&Monitor->m_SensorDataMutex);
118:

```

```

119:         //
120:         // If no reset, make sure the sensor is still connected
121:         //
122:         pthread_mutex_lock(&Monitor->m_SensorDataMutex);
123:         if (!ThreadReset && Monitor->m_SensorData[i].Registered())
124:         {
125: #ifdef DEBUG_MONITOR
126:             printf("Checking connection... \n");
127: #endif
128:             if (!Monitor->m_Sensors[i].CheckConnection())
129:             {
130: #ifdef DEBUG_MONITOR
131:                 printf("Sensor disconnect, restarting. \n");
132: #endif
133:                 Monitor->m_Sensors[i].CancelSensor(true);
134:                 Monitor->m_SensorData[i].Registered(false);
135:                 Monitor->m_Sensors[i].StartUSBMonitor(i, Monitor);
136:             }
137:         }
138:         pthread_mutex_unlock(&Monitor->m_SensorDataMutex);
139:
140:
141:         //
142:         // Check for sensors who have not been acked after SDMHHello msgs have been sent
143:         //
144:         pthread_mutex_lock(&Monitor->m_SensorDataMutex);
145:         if (Monitor->m_SensorData[i].Registered() && !Monitor->m_SensorData[i].Acked())
146:         {
147:             CurSeconds = GetCurSeconds();
148:             // Check for a hello-ack timeout
149:             if (Monitor->m_SensorData[i].CheckAckTimeout(CurSeconds))
150:             {
151: #ifdef DEBUG_MONITOR
152:                 printf("SDMAck not received after SDMHHello, resending Hello for sensorIndex:
%i. \n", i);
153: #endif
154:                 SDMHHello helloMsg;
155:                 helloMsg.type = 'D';
156:                 helloMsg.source.setPort(PORT_SM);
157:                 helloMsg.source.setSensorID(i);
158:                 helloMsg.Send();

```

```

159:         }
160:     }
161:     pthread_mutex_unlock(&Monitor->m_SensorDataMutex);
162:
163:     //
164:     // Check for sensors who have been sent their xTEDS but not yet received confirmation
of registration
165:     //
166:     pthread_mutex_lock(&Monitor->m_SensorDataMutex);
167:     if (Monitor->m_SensorData[i].Acked() && !Monitor-
>m_SensorData[i].RegConfirmed())
168:     {
169:         CurSeconds = GetCurSeconds();
170:         // Check for a xteds-id timeout
171:         if (Monitor->m_SensorData[i].CheckIDTimeout(CurSeconds))
172:         {
173: #ifdef DEBUG_MONITOR
174:             printf("SDMID not received after SDMxTEDS, resending xTEDS for
sensorIndex: %i. \n", i);
175: #endif
176:             Monitor->SendxTEDS(i);
177:         }
178:     }
179:     pthread_mutex_unlock(&Monitor->m_SensorDataMutex);
180:
181: }
182: sleep(SENSOR_CHECK_INTERVAL);
183: }
184: return NULL;
185: }
186:
187: bool SensorMonitor::SensorRegistering(int SensorNum)
188: {
189:     if (SensorNum < 0 || SensorNum >= m_NumSensors)
190:         return false;
191:
192:     // Set the new registration start time
193:     pthread_mutex_lock(&m_SensorDataMutex);
194:     if (m_SensorData[SensorNum].Registered())
195:     {
196:         pthread_mutex_unlock(&m_SensorDataMutex);
197:         return true;

```

```

198:     }
199:     m_SensorData[SensorNum].SetStartTime(GetCurSeconds());
200:     pthread_mutex_unlock(&m_SensorDataMutex);
201:
202:     return true;
203: }
204:
205: bool SensorMonitor::SensorRegistered(int SensorNum)
206: {
207:     if (SensorNum < 0 || SensorNum >= m_NumSensors)
208:         return false;
209:
210:     // Set the sensor as registered and reset its start time
211:     pthread_mutex_lock(&m_SensorDataMutex);
212:     m_SensorData[SensorNum].Registered(true);
213:     m_SensorData[SensorNum].SetStartTime(SECONDS_INIT_VAL);
214:     pthread_mutex_unlock(&m_SensorDataMutex);
215:
216:     return true;
217: }
218:
219: bool SensorMonitor::SensorAcked(int SensorNum)
220: {
221:     if (SensorNum < 0)
222:         return false;
223:
224:     // Set the sensor as acked
225:     pthread_mutex_lock(&m_SensorDataMutex);
226:     if(m_SensorData[SensorNum].GetHelloTime() != 0)
227:     {
228:         m_SensorData[SensorNum].Acked(true);
229:     }
230:     pthread_mutex_unlock(&m_SensorDataMutex);
231:
232:     return true;
233: }
234:
235: bool SensorMonitor::SensorRegConfirmed(int SensorNum)
236: {
237:     if (SensorNum < 0)
238:         return false;

```

```

239:
240: // Set the sensor as acked
241: pthread_mutex_lock(&m_SensorDataMutex);
242: m_SensorData[SensorNum].RegConfirmed(true);
243: pthread_mutex_unlock(&m_SensorDataMutex);
244:
245: return true;
246: }
247:
248: unsigned int SensorMonitor::SetHelloSendTime(int SensorNum)
249: {
250: if (SensorNum < 0)
251: {
252:     return 0;
253: }
254: // Set the time the Hello was sent
255: unsigned int curSecs = GetCurSeconds();
256:
257: pthread_mutex_lock(&m_SensorDataMutex);
258: m_SensorData[SensorNum].SetHelloTime(curSecs);
259: pthread_mutex_unlock(&m_SensorDataMutex);
260:
261: return curSecs;
262: }
263:
264: unsigned int SensorMonitor::SetxTEDSSendTime(int SensorNum)
265: {
266: if (SensorNum < 0)
267:     return 0;
268:
269: // Set the time thge xTEDS was sent
270: unsigned int curSecs = GetCurSeconds();
271: pthread_mutex_lock(&m_SensorDataMutex);
272: m_SensorData[SensorNum].SetxTEDSTime(curSecs);
273: pthread_mutex_unlock(&m_SensorDataMutex);
274:
275: return curSecs;
276: }
277:
278: bool SensorMonitor::SendxTEDS(int SensorNum)
279: {

```



```

280: if (SensorNum < 0)
281: {
282:     return false;
283: }
284:
285: SDMxTEDS xTEDSMsg;
286: strcpy(xTEDSMsg.xTEDS, m_Sensors[SensorNum].device_xTEDS);
287: xTEDSMsg.source = m_Sensors[SensorNum].sensor_id;
288: strcpy(xTEDSMsg.SPA_node, m_Sensors[SensorNum].device.USBLocation());
289: xTEDSMsg.Send();
290: MessageSent(&xTEDSMsg);
291:
292: // Set the time thge xTEDS was sent
293: //pthread_mutex_lock(&m_SensorDataMutex);
294: unsigned int sentTime = SetxTEDSSendTime(SensorNum);
295: //pthread_mutex_unlock(&m_SensorDataMutex);
296: #ifdef DEBUG_MONITOR
297: printf("xTEDS for sensor index: %i sent at: %i \n", SensorNum, sentTime);
298: #endif
299:
300: return true;
301: }
302:
303: Sensor& SensorMonitor::operator [] (int Index)
304: {
305:     return m_Sensors[Index];
306: }
307:

```

File: sdm/sm/sm.cpp

```
1: //*****
2: //Sensor Manager Module of the SDM
3: //
4: //This source is compilable on both a Linux and a Win32 architecture. Throughout, there are
5: //compile switches as #ifdef ... #endif which conditionally add or remove functionality.
6: //The most notable changes for compilation is the removal of a separate signal handling thread
7: //under Win32, because Windows has far fewer options for process signals than Linux.
8: //*****
9:
10: #include "sm.h"
11: #include "SensorMonitor.h"
12:
13:
14: unsigned int MAX_SENSORS=15;
15: //Sensor* sensor;
16: SensorMonitor sensor;
17:
18: SubscriptionManager sm_subscriptions;
19: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER;
20: pthread_mutex_t perf_counter_mutex = PTHREAD_MUTEX_INITIALIZER;    //a    mutex    for
performance counter access
21: pthread_mutex_t dm_found_mutex = PTHREAD_MUTEX_INITIALIZER;        //a    mutex    to
cooperate the verifyDM procedure
22: pthread_mutex_t log_service_mutex = PTHREAD_MUTEX_INITIALIZER;    //a    mutex    for the
message logger service
23: // Locked before any sensor write, this is another concurrency bug in the ASIM driver
24: // using a mutex for all requests prevents messages from stepping on each others' toes
25: // in the driver
26: pthread_mutex_t sensor_write_mutex = PTHREAD_MUTEX_INITIALIZER;
27: unsigned long Address_SM = inet_addr("127.0.0.1");
28: unsigned int total_rcd = 0;    //message counter for total received for life of sm
29: unsigned int prevsec_rcd = 0;    //message counter for total received previous second
30: unsigned int total_sent = 0;    //message counter for total sent for life of sm
31: unsigned int prevsec_sent = 0;    //message counter for total sent previous second
32: bool dm_found = false;
33: int debug;
34: bool prompt;
35: MessageLogger log_service;
36: short tm_mode_msg_id = 0;
37: unsigned char tm_mode;
```

```

38: SDMComponent_ID tm_mode_source;
39:
40: bool ackReceived = false;
41: bool registerReceived = false;
42: bool idReceived = false;
43:
44: char * sm_xTEDS = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n<xTEDS version= \"2.0 \"
xmlns=
    \"http://www.interfacecontrol.com/SPA/xTEDS
    \"
    xmlns:xsi=
    \"http://www.w3.org/2001/XMLSchema-instance
    \"
    xsi:schemaLocation=
    \"http://www.interfacecontrol.com/SPA/xTEDS
    ../Schema/xTEDS02.xsd
    \"
    name=
    \"Sensor_Manager_xTEDS \"/> \n \t<Application kind= \"Software \" name= \"SensorManager \"/> \n
\t<Interface name= \"Msg_Count \" id= \"2 \"/> \n \t \t<Variable name= \"Total_Messages_Recd \" kind=
\"Total \" format= \"UINT32 \"/> \n \t \t<Variable name= \"Messages_Last_Second_Recd \" kind=
\"Total \" format= \"UINT32 \"/> \n \t \t<Variable name= \"Total_Messages_Sent \" kind= \"Total \"
format= \"UINT32 \"/> \n \t \t<Variable name= \"Messages_Last_Second_Sent \" kind= \"Total \"
format= \"UINT32 \"/> \n \n \t \t<Notification> \n \t \t \t<DataMsg name= \"Message_Count \" id= \"13 \"
msgArrival= \"PERIODIC \"/> \n \t \t \t \t<VariableRef name= \"Total_Messages_Recd \"/> \n \t \t \t \t
\t<VariableRef name= \"Messages_Last_Second_Recd \"/> \n \t \t \t \t \t<VariableRef name=
\"Total_Messages_Sent \"/> \n \t \t \t \t \t<VariableRef name= \"Messages_Last_Second_Sent \"/> \n \t \t
\t</DataMsg> \n \t \t</Notification> \n \t</Interface> \n \t<Interface name= \"Message_Log \" id= \"3 \"/>
\n \t \t<Variable format= \"UINT08 \" kind= \"TBD \" name= \"Msg_Type \"/> \n \t \t<Variable format=
\"UINT32 \" kind= \"IP_long \" name= \"Address \"/> \n \t \t<Variable format= \"UINT16 \" kind=
\"Port_of_Device \" name= \"Port \"/> \n \t \t<Variable format= \"UINT32 \" kind= \"ID \" name=
\"Sensor_ID \"/> \n \t \t<Command> \n \t \t \t<CommandMsg name= \"Enable_Logging \" id= \"16 \"/> \n
\t \t \t \t<VariableRef name= \"Msg_Type \"/> \n \t \t \t \t<VariableRef name= \"Address \"/> \n \t \t \t
\t<VariableRef name= \"Port \"/> \n \t \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \t \t \t
\t</CommandMsg> \n \t \t</Command> \n \t \t<Command> \n \t \t \t<CommandMsg name=
\"Disable_Logging \" id= \"17 \"/> \n \t \t \t \t<VariableRef name= \"Msg_Type \"/> \n \t \t \t \t
\t<VariableRef name= \"Address \"/> \n \t \t \t \t \t<VariableRef name= \"Port \"/> \n \t \t \t \t
\t<VariableRef name= \"Sensor_ID \"/> \n \t \t \t</CommandMsg> \n \t \t</Command> \n \t</Interface> \n</xTEDS>";
45:
46: int main(int argc, char** argv)
47: {
48: int result;
49: SDM_TimeInit();
50: static struct option long_options[] = {
51:     {"dm",1,0,'d'},
52:     {"help",0,0,'h'},
53:     {"max_sensors",1,0,'n'},
54:     {"debug",1,0,'g'},
55:     {"prompt",0,0,'p'},
56:     {0,0,0,0}
57: };
58: int option_index;
59: int option;

```

```

60: pthread_t UDP_thread;
61: pthread_t TCP_thread;
62: pthread_t SigHandler_thread;
63: bool DM_set = false;
64: prompt = false;
65: SDMxTEDS xteds;
66: #ifndef WIN32
67: sigset_t signal_set;
68: #endif
69: SDMReqReg reqreg_msg;
70:
71: //parse command line options
72: while(1)
73: {
74:     option = getopt_long(argc,argv,"g:hn:d:p", long_options, &option_index);
75:     if(option== -1)
76:         break; //no more options
77:     switch(option)
78:     {
79:         case 'h':
80:             printf("Usage: sm [options] \n");
81:             printf("Options: \n");
82:             printf("--dm=<addr> -d<addr> \t \t \tSet IP address of Data Manager \n \t \t \t \t<addr>
should be given in dot number notation \n");
83:             printf("--help -h \t \t \tDisplay this information \n");
84:             printf("--max_sensors=<number> -n<number> \tSet maximum <number> of ASIMs for
this Sensor Manager \n");
85:             printf("--debug=<level> -g<level> \t \tSet level of debug messages \n \t \t \t \t0=none,
1=moderate, 2=verbose \n");
86:             printf("--prompt -p \t \t \tPrompt for each sensor before monitoring \n");
87:             return 0;
88:         case 'd':
89:             DataManager.setAddress(inet_addr(optarg));
90:             if (((long)DataManager.getAddress()) == -1)
91:             {
92:                 printf("Error in DM address. Be sure to use --dm= instead of -dm= \n");
93:                 return 0;
94:             }
95:             DataManager.setPort(PORT_DM);
96:             DM_set = true;
97:             break;
98:         case 'n':

```

```

99:         MAX_SENSORS = atoi(optarg);
100:         break;
101:     case 'g':
102:         debug = atoi(optarg);
103:         break;
104:     case 'p':
105:         prompt = true;
106:         break;
107:     case '?':
108:         break;
109:     }
110: }
111: // Get the address of this SensorManager node
112: if (GetNodeAddress() == -1)
113:     printf("Unable to get the SM's IP address, using localhost instead. \n");
114:
115: // If the DataManager address wasn't specified on the command line, assume localhost
116: if(!DM_set)
117: {
118:     DataManager.setAddress(inet_addr("127.0.0.1"));
119:     DataManager.setPort(PORT_DM);
120: }
121: debug_f(2,"Finding DM at address 0x%lx \n",DataManager.getAddress());
122: printf("SM %s (%d sensors) \n",SDM_VERSION,MAX_SENSORS);
123: switch(debug)
124: {
125:     case 1:
126:         printf("SM in debug 1 (moderate). \n");
127:         break;
128:     case 2:
129:         printf("SM in debug 2 (verbose). \n");
130:         break;
131:     case 3:
132:         printf("SM in debug 3 (verbose w/msg echo). \n");
133:         break;
134:     }
135:
136: //All subsequent threads block the SIGINT and SIGALRM signals so they aren't interrupted after
calling pthread_mutex_lock()
137: //this avoids a deadlock situation by dedicating a single thread for signal handling.
138: #ifdef WIN32

```

```

139:  signal(SIGINT,SigIntHandler);
140: #else
141:  sigemptyset(&signal_set);
142:  sigaddset(&signal_set, SIGALRM);
143:  sigaddset(&signal_set, SIGINT);
144:  pthread_sigmask(SIG_BLOCK, &signal_set, NULL);
145: #endif
146:
147:  //Start listener threads, then verify DM
148:  result = pthread_create(&UDP_thread,NULL,&UDPMonitor,NULL);
149:  if(result < 0)
150:  {
151:      perror("Could not start UDP Listener");
152:  }
153:  result = pthread_create(&TCP_thread,NULL,&TCPMonitor,NULL);
154:  if(result < 0)
155:  {
156:      perror("Could not start TCP Listener");
157:  }
158:  /*No signal handling thread for Win32*/
159: #ifndef WIN32
160:  result = pthread_create(&SigHandler_thread,NULL,&SigHandler,NULL);
161:  if(result < 0)
162:  {
163:      perror("Could not start Sig Handler thread.");
164:  }
165: #endif
166:  printf("Listening for UDP messages on port %d \n",PORT_SM);
167:  printf("Listening for TCP message on port %d \n",PORT_SM);
168:  fflush(NULL);
169:  VerifyDM();
170:
171:
172:  SDMHHello hello;
173:  hello.source.setPort(PORT_SM);
174:  hello.type = 'C';
175:  double endTime = 0;
176:  double timeOut = 5.0;
177:  while(!ackReceived)
178:  {
179:      if(GetCurTime() > endTime)

```

```

180:  {
181:      debug_f(1, "Sending Hello \n");
182:      hello.Send();
183:      endTime = GetCurTime() + timeOut;
184:  }
185:  usleep(10000);
186:  }
187:  while(!registerReceived)
188:  {
189:      usleep(10000);
190:  }
191:  debug_f(1, "Registering xTEDS \n");
192:
193:  // Send the SensorManager's xTEDS
194:  xteds.source.setSensorID(1);
195:  xteds.source.setPort(PORT_SM);
196:  strcpy(xteds.xTEDS, sm_xTEDS);
197:  xteds.Send();
198:  MessageSent(&xteds);
199:
200:  while (!idReceived)
201:  {
202:      usleep(10000);
203:  }
204:  debug_f(1, "SDMID received \n");
205:
206:  //Send ReqReg for TM's Mode notification
207:  reqreg_msg.destination.setPort(PORT_SM);
208:  reqreg_msg.reply = SDM_REQREG_CURRENT_AND_FUTURE;
209:  reqreg_msg.id = TM_Mode_ID;
210:  strcpy(reqreg_msg.device, "TaskManager");
211:  strcpy(reqreg_msg.interface, "TM_Interface");
212:  strcpy(reqreg_msg.item_name, "Status");
213:  reqreg_msg.Send();
214:
215:  //initialize sensor array
216:  if (!sensor.InitializeSensors(MAX_SENSORS, prompt, debug))
217:  {
218:      printf("Error initializing sensors. \n");
219:      return 0;
220:  }

```

```

221:
222:  pthread_join(UDP_thread,NULL);
223:  pthread_join(TCP_thread,NULL);
224: }
225:
226:
227: double GetCurTime()
228: {
229:  unsigned int seconds;
230:  unsigned int uSeconds;
231:  double curTime;
232:  SDM_GetTime(&seconds, &uSeconds);
233:  curTime = seconds + ((double)uSeconds/1000000.0);
234:  return curTime;
235: }
236:
237:
238:
239: #ifdef WIN32
240: /* SigIntHandler for Win32 handles only SIGINT. Upon a SIGINT, the SensorManager cancels its
xTEDS and the xTEDS of any connected
241:  devices, then the process exits.
242:  INPUTS:
243:      signum - The signal number returned by the system.
244:  RETURNS:
245:      void
246: */
247: void SigIntHandler(int signum)
248: {
249:  SDMCancelxTEDS cancel;
250:  SDMCancel cancel_msg;
251:  unsigned int i;
252:  int sig;
253: #ifndef WIN32
254:  sigset_t signal_set;
255: #endif
256:  cancel.source.setPort(PORT_SM);
257:  if (signum == SIGINT)
258:  {
259:      printf(" \nShutting down \n");
260:      pthread_mutex_lock(&dm_found_mutex);

```



```

261:     if (dm_found)
262:     {
263:         // Cancel each sensor
264:         for(i=0;i<MAX_SENSORS;++i)
265:         {
266:             cancel.source.setSensorID(i+SM_OFFSET);
267:             cancel.Send();
268:             MessageSent(&cancel);
269:         }
270:         // Cancel the SensorManager
271:         cancel.source.setSensorID(1);
272:         cancel.Send();
273:         MessageSent(&cancel);
274:     }
275:     pthread_mutex_unlock(&dm_found_mutex);
276:     // Cancel the subscription to the TM mode message
277:     cancel_msg.source = tm_mode_source;
278:     cancel_msg.destination.setPort(PORT_SM);
279:     cancel_msg.msg_id = tm_mode_msg_id;
280:     cancel_msg.Send();
281:     MessageSent(&cancel_msg);
282:     exit(EXIT_SUCCESS);
283: }
284: }
285: #else
286: /*
287:  Signal handler for the Linux implementation handles both SIGINT and SIGALRM.  SIGINT will
288:  cause the SensorManager to cancel its xTEDS
289:  and the xTEDS of any connected devices, then exit the process.  SIGALRM will publish the
290:  performance counter message and reset the
291:  second-count performance counters back to zero.  This signal handler is a signal handling thread
292:  which prevents deadlock because
293:  the SIGALRM section locks a mutex.
294:  INPUTS:
295:  arg - Not used, only there to match the function signature of thread routines.
296:  RETURNS:
297:  void * - Always NULL.
298:  */
299: void* SigHandler(void* arg)
300: {
301:     SDMCancelxTEDS cancel;
302:     SDMCancel cancel_msg;

```

```

300: unsigned int i;
301: int sig;
302: sigset_t signal_set;
303: cancel.source.setPort(PORT_SM);
304: while (1)
305: {
306:     sigfillset(&signal_set);
307:     sigwait(&signal_set, &sig);
308:
309:     switch(sig)
310:     {
311:     case SIGINT:
312:         // Cancel all xTEDS and shutdown
313:         printf(" \nShutting down \n");
314:         pthread_mutex_lock(&dm_found_mutex);
315:         if (dm_found)
316:         {
317:             // Cancel each sensor
318:             for(i=0;i<MAX_SENSORS;++i)
319:             {
320:                 cancel.source.setSensorID(i+SM_OFFSET);
321:                 cancel.Send();
322:                 MessageSent(&cancel);
323:             }
324:             // Cancel the SensorManager
325:             cancel.source.setSensorID(1);
326:             cancel.Send();
327:             MessageSent(&cancel);
328:         }
329:         pthread_mutex_unlock(&dm_found_mutex);
330:         // delete [] sensor;
331:         // Cancel the subscription to the TM mode message
332:         cancel_msg.source = tm_mode_source;
333:         cancel_msg.destination.setPort(PORT_SM);
334:         cancel_msg.msg_id = tm_mode_msg_id;
335:         cancel_msg.Send();
336:         MessageSent(&cancel_msg);
337:         exit(EXIT_SUCCESS);
338:         break;
339:     case SIGALRM:
340:         // Publish performance counters and reset to zero

```

```

341:         PublishPerformanceCounterMessage();
342:         pthread_mutex_lock(&perf_counter_mutex);
343:         prevsec_recd = 0;
344:         prevsec_sent = 0;
345:         pthread_mutex_unlock(&perf_counter_mutex);
346:     }
347: }
348: }
349: #endif
350:
351:
352:
353: void* AckHandler(void* arg)
354: {
355:     SDMAck ackMsg;
356:     char* buf = (char*)arg;
357:     if (ackMsg.Unmarshal(buf) < 0)
358:     {
359:         printf("Invalid SDMAck message. \n");
360:         delete [] buf;
361:         return NULL;
362:     }
363:     // Log message
364:     MessageReceived(&ackMsg);
365:
366:     if(!ackReceived)
367:     {
368:         debug_f(1, "Ack received for the SM \n");
369:         ackReceived = true;
370:         return NULL;
371:     }
372:
373:     debug_f(1, "Ack received for sensor index: %i \n", ackMsg.error);
374:     if(ackMsg.error >= 0)
375:     {
376:         sensor.SensorAcked(ackMsg.error);
377:     }
378:
379:     return NULL;
380: }
381:

```

```

382:
383: void* RegisterHandler(void* arg)
384: {
385:     SDMRegister registerMsg;
386:     char* buf = (char*)arg;
387:     if (registerMsg.Unmarshal(buf) < 0)
388:     {
389:         printf("Invalid SDMRegister message. \n");
390:         delete [] buf;
391:         return NULL;
392:     }
393:     // Log message
394:     MessageReceived(&registerMsg);
395:
396:     if(!registerReceived)
397:     {
398:         debug_f(1, "Register msg received for the SM \n");
399:         registerReceived = true;
400:         return NULL;
401:     }
402:
403:     debug_f(1, "Register msg received for sensor index: %hi \n", registerMsg.sensorIndex);
404:     if(registerMsg.sensorIndex >= 0)
405:     {
406:         sensor.SendxTEDS(registerMsg.sensorIndex);
407:     }
408:
409:     return NULL;
410: }
411:
412: void* IDHandler(void* arg)
413: {
414:     SDMID idMsg;
415:     char* buf = (char*)arg;
416:     if (idMsg.Unmarshal(buf) < 0)
417:     {
418:         printf("Invalid SDMID message. \n");
419:         delete [] buf;
420:         return NULL;
421:     }
422:     // Log message

```

```

423:   MessageReceived(&idMsg);
424:
425:   if(!idReceived)
426:   {
427:       debug_f(1, "ID msg received for the SM \n");
428:       idReceived = true;
429:       return NULL;
430:   }
431:
432:       debug_f(1, "ID msg received for sensor index: %i \n",
(int)((idMsg.destination.getSensorID()&0x00FF) - SM_OFFSET));
433:   if(idMsg.destination.getSensorID() != 0)
434:   {
435:       unsigned long sensorIndex = (idMsg.destination.getSensorID()&0x00FF) - SM_OFFSET;
436:       sensor.SensorRegConfirmed(sensorIndex);
437:   }
438:
439:   return NULL;
440: }
441:
442:
443: /*
444:   ServiceHandler handles the receipt of SDMSERVICE messages. These messages are forwarded to
the appropriate ASIM based on the SensorID
445:   number of the source component identifier in the message.
446:   INPUTS:
447:       arg - The buffer containing the SDMSERVICE message.
448:   RETURNS:
449:       void * - Always NULL
450: */
451: void* ServiceHandler(void* arg)
452: {
453:   SDMSerreqst service_request;
454:   char* buf = (char*)arg;
455:   if (service_request.Unmarshal(buf) < 0)
456:   {
457:       printf("Invalid SDMSERVICE message. \n");
458:       delete [] buf;
459:       return NULL;
460:   }
461:   // Log message

```

```

462: MessageReceived(&service_request);
463:
464: // Request sensor identifier of the sensor on which to perform the service
465: const unsigned long request_sid = (service_request.source.getSensorID()&0x00FF);
466: debug_f(1,"Service request for sensor id %lu => asim%lu\n",service_request.source.getSensorID(),request_sid-SM_OFFSET);
467:
468: // Determine which sensor should get this message
469: // If the request is within a valid range
470: if (request_sid-SM_OFFSET < MAX_SENSORS)
471:     sensor[request_sid-SM_OFFSET].Service(service_request);
472: else
473:     printf("Error: Service request for sensor; sensor id out of range (%lu) \n",request_sid);
474:     delete [] buf;
475:     return NULL;
476: }
477: /*
478: CommandHandler handles the receipt of SDMCommand messages. These messages are
forwarded to the appropriate ASIM based on the SensorID
479: number of the source component identifier in the message. If the SensorID applies to the
SensorManager, then the appropriate command
480: is performed based on the command messages defined in the SM's xTEDS.
481: INPUTS:
482:     arg - The buffer containing the SDMCommand message.
483: RETURNS:
484:     void * - Always NULL.
485: */
486: void* CommandHandler(void* arg)
487: {
488:     SDMCommand command;
489:     char* buf = (char*)arg;
490:     // Unmarshal message
491:     if (command.Unmarshal(buf) < 0)
492:     {
493:         printf("Invalid SDMCommand message. \n");
494:         delete [] buf;
495:         return NULL;
496:     }
497:     // Log message
498:     MessageReceived(&command);
499:     // Request sensor identifier of the sensor on which to perform the command
500:     const unsigned long request_sid = (command.source.getSensorID()&0x00FF);

```

```

501:  debug_f(1,"Command      for      sensor      id      %lu      ==>      asim%lu
\n",command.source.getSensorID(),request_sid-SM_OFFSET);
502:
503:  // Determine which sensor should get this message
504:  // Command for the SensorManager (Enable message logging)
505:  if (request_sid == SENSOR_MANAGER_SID && command.command_id ==
CMD_ENABLE_LOGGING)
506:  {
507:      pthread_mutex_lock(&log_service_mutex);
508:      if (log_service.NeedsInit())
509:          log_service.SetLogFile("Sensor Manager Message Log \n","smmessages.log");
510:      log_service.AddMessageType(&command);
511:      pthread_mutex_unlock(&log_service_mutex);
512:  }
513:  // Command for the SensorManager (Disable message logging)
514:  else if (request_sid == SENSOR_MANAGER_SID && command.command_id ==
CMD_DISABLE_LOGGING)
515:  {
516:      pthread_mutex_lock(&log_service_mutex);
517:      log_service.RemoveMessageType(&command);
518:      pthread_mutex_unlock(&log_service_mutex);
519:  }
520:  // Commmand for an ASIM, if within valid range
521:  else if (request_sid != SENSOR_MANAGER_SID && request_sid-SM_OFFSET <
MAX_SENSORS)
522:  {
523:      pthread_mutex_lock(&sensor_write_mutex);
524:      usleep(1500);
525:      sensor[request_sid-SM_OFFSET].Command(command);
526:      pthread_mutex_unlock(&sensor_write_mutex);
527:  }
528:  else
529:  {
530:      printf("Error: command request for sensor; no matching command or sensor id out of range
(%lu). \n",request_sid);
531:  }
532:  delete [] buf;
533:  return NULL;
534: }
535: /*
536:  SubscriptionHandler handles the receipt of SDMSubreqst messages.  If the message is for an
ASIM, it is forwarded appropriately based on the

```

```

537:   SensorID in the source component identifier of the message.  If the message is for the
SensorManager, the requester is added to the SM's
538:   subscription list based on the messages defined in the SM's xTEDS.
539:   INPUTS:
540:       arg - The buffer containing the SDMSubreqst message.
541:   RETURNS:
542:       void * - Always NULL.
543: */
544: void* SubscriptionHandler(void* arg)
545: {
546:     SDMSubreqst request;
547:     char* buf = (char*)arg;
548:     // Unmarshal message
549:     if (request.Unmarshal(buf) < 0)
550:     {
551:         printf("Invalid SDMSubreqst message. \n");
552:         delete [] buf;
553:         return NULL;
554:     }
555:     // Log message
556:     MessageReceived(&request);
557:     // Request sensor identifier of the sensor on which to perform the subscription request
558:     const unsigned long request_sid = (request.source.getSensorID() & 0x00FF);
559:     debug_f(1, "Subscription Request for sensor id %ld ==> asim%lu\n", request.source.getSensorID(), request_sid - SM_OFFSET);
560:
561:     // Determine which sensor should get this message
562:     // Performance counter subscription request for the SensorManager
563:     if (request_sid == SENSOR_MANAGER_SID && request.msg_id ==
SUB_PERFORMANCE_COUNTERS)
564:     {
565: #ifndef WIN32
566:         fflush(NULL);
567:         pthread_mutex_lock(&subscription_mutex);
568:         sm_subscriptions.AddSubscription(request);
569:         pthread_mutex_unlock(&subscription_mutex);
570:         itimerval interval;
571:         getitimer(ITIMER_REAL, &interval);
572:         if (interval.it_interval.tv_sec == 0 && interval.it_value.tv_sec == 0)
573:         {
574:             //Time interval for the publish interval of the performance counter
575:             timeval pubInterval;

```



```

576:         pubInterval.tv_sec = 1;
577:         pubInterval.tv_usec = 0;
578:
579:         itimerval timerInterval;
580:         timerInterval.it_interval = pubInterval;
581:         timerInterval.it_value = pubInterval;
582:
583:         //Set the performance counter timer
584:         setitimer(ITIMER_REAL, &timerInterval, NULL);
585:     }
586: #endif
587: }
588: // Subscription request for an ASIM, if within valid range
589: else if (request_sid != 1 && request_sid-SM_OFFSET < MAX_SENSORS)
590: {
591:     pthread_mutex_lock(&sensor_write_mutex);
592:     usleep(1500);
593:     sensor[request_sid-SM_OFFSET].Subscribe(request);
594:     pthread_mutex_unlock(&sensor_write_mutex);
595: }
596: else
597: {
598:     printf("Error: subscription request for sensor; no matching subscription or sensor id out of
range (%lu). \n",request_sid);
599: }
600: delete [] buf;
601: return NULL;
602: }
603: /*
604: DeleteSubscriptionHandler handles the receipt of SDMDeletesub messages. If the request is for
an ASIM, it is forwarded appropriately based
605: on the SensorID component identifier. If the request is for the SensorManager, the subscription is
removed from the SM's subscription list
606: based on the message definitions in its xTEDS.
607: INPUTS:
608:     arg - The buffer containing the SDMDeletesub message.
609: RETURNS:
610:     void * - Always NULL.
611: */
612: void* DeleteSubscriptionHandler(void* arg)
613: {
614:     SDMDeletesub cancel;

```

```

615:  char* buf = (char*)arg;
616:  // Unmarshal message
617:  if (cancel.Unmarshal(buf) < 0)
618:  {
619:      printf("Invalid SDMDeletesub message. \n");
620:      delete [] buf;
621:      return NULL;
622:  }
623:  // Log message
624:  MessageReceived(&cancel);
625:
626:  // Get the sensor identifier of the sensor on which to perform the delete subscription request
627:  const unsigned long request_sid = (cancel.source.getSensorID()&0x00FF);
628:  debug_f(1,"Cancel      Request      for      sensor      id      %lu      =>      asim%lu
\n",cancel.source.getSensorID(),request_sid-SM_OFFSET);
629:
630:  // Determine which sensor should get this message
631:  // If this is a request for the SensorManager
632:  if      (request_sid      ==      SENSOR_MANAGER_SID      &&      cancel.msg_id      ==
SUB_PERFORMANCE_COUNTERS)
633:  {
634:      pthread_mutex_lock(&subscription_mutex);
635:      sm_subscriptions.RemoveSubscription(cancel);
636:      pthread_mutex_unlock(&subscription_mutex);
637:  }
638:  // Request for ASIM, check to be sure that the request is within a valid range
639:  else if (request_sid != SENSOR_MANAGER_SID && request_sid-SM_OFFSET <
MAX_SENSORS)
640:  {
641:      pthread_mutex_lock(&sensor_write_mutex);
642:      usleep(1500);
643:      sensor[request_sid-SM_OFFSET].CancelSubscription(cancel);
644:      pthread_mutex_unlock(&sensor_write_mutex);
645:  }
646:  // Error
647:  else
648:  {
649:      printf("Error: delete subscription request; no matching subscription or sensor id out of range
(%lu). \n",request_sid);
650:  }
651:  delete [] buf;
652:  return NULL;

```

```

653: }
654: /*
655:    TatHandler handles the receipt of SDMTat messages and forwards the message down to ASIMs.
    If the SensorID field is zero, the message is
656:    forwarded to all ASIMs, otherwise the message is sent to the ASIM specified.
657:    INPUTS:
658:        arg - The buffer containing the SDMTat message.
659:    RETURNS:
660:        void * - Always NULL.
661: */
662: void* TatHandler(void* arg)
663: {
664:     SDMTat tat;
665:     char* buf = (char*)arg;
666:     // Unmarshal message
667:     if (tat.Unmarshal(buf) < 0)
668:     {
669:         printf("Invalid SDMTat message. \n");
670:         delete [] buf;
671:         return NULL;
672:     }
673:     // Log message
674:     MessageReceived(&tat);
675:
676:     // Request sensor identifier of the sensor on which to perform the tat
677:     const unsigned long request_sid = (tat.destination.getSensorID() & 0x00FF);
678:
679:     if(tat.destination.getSensorID() == 0) //send to all sensors
680:     {
681:         debug_f(1, "Time at Tone Request for all sensors \n");
682:
683:         for(unsigned int i=0; i<MAX_SENSORS; ++i)
684:         {
685:             pthread_mutex_lock(&sensor_write_mutex);
686:             usleep(1500);
687:             sensor[i].TimeAtTone(tat.seconds, tat.useconds);
688:             pthread_mutex_unlock(&sensor_write_mutex);
689:         }
690:     }
691:     else
692:     {

```

```

693:     debug_f(1,"Time at Tone Request for sensor id %lu => asim%lu
\n",tat.destination.getSensorID(),request_sid-SM_OFFSET);
694:
695:     //determine which sensor should get this message
696:     // Make sure request is within valid range
697:     if (request_sid-SM_OFFSET < MAX_SENSORS)
698:     {
699:         pthread_mutex_lock(&sensor_write_mutex);
700:         usleep(1500);
701:         sensor[request_sid-SM_OFFSET].TimeAtTone(tat.seconds,tat.useconds);
702:         pthread_mutex_unlock(&sensor_write_mutex);
703:     }
704:     else
705:         printf("Error: time-at-tone request; sensor id out of range (%lu). \n",request_sid);
706: }
707: delete [] buf;
708: return NULL;
709: }
710: /*
711: ReadyHandler handles the receipt of SDMReady message from the DataManager. This routine
handles the replies from the VerifyDM routine which
712: verifies the existence of the DataManager. This routine is also used to get the IP address of the
DM.
713: INPUTS:
714:     arg - The buffer containing the SDMReady message.
715: RETURNS:
716:     void * - Always NULL.
717: */
718: void* ReadyHandler(void* arg)
719: {
720:     SDMReady ready;
721:     char* buf = (char*)arg;
722:     bool m_dm_found = false;
723:     pthread_mutex_lock(&dm_found_mutex);
724:     m_dm_found = dm_found;
725:     pthread_mutex_unlock(&dm_found_mutex);
726:
727:     // Unmarshal the message
728:     if (ready.Unmarshal(buf) < 0)
729:     {
730:         printf("Received invalid ready message. \n");
731:         delete [] buf;

```

```

732:     return NULL;
733: }
734: else
735:     MessageReceived(&ready); // Log message
736:
737: // If this is a heartbeat request, respond
738: if ((ready.source.getPort() == PORT_SM_MONITOR) && (ready.source.getAddress() ==
inet_addr("127.0.0.1"))) && (ready.source.getSensorID() == 0))
739: {
740:     debug_f(3,"Heartbeat message received. \n");
741:     ready.SendTo(ready.source);
742:     MessageSent(&ready);
743:     delete [] buf;
744:     return NULL;
745: }
746: // If the DataManager has not yet been found, this is a response from it
747: if (!m_dm_found)
748: {
749:     if (ready.destination.getPort() == PORT_DM && ready.source.getPort() == PORT_SM)
750:     {
751:         DataManager = ready.destination;
752:         // Set DM found flag to true
753:         pthread_mutex_lock(&dm_found_mutex);
754:         dm_found = true;
755:         pthread_mutex_unlock(&dm_found_mutex);
756:     }
757: }
758: delete [] buf;
759: return NULL;
760: }
761: /*
762:  RegInfoHandler handles the receipt of SDMRegInfo message to the SensorManager. The SM
issues an SDMReqReg for the TaskManager's mode
763:  message. The mode message will be used to reset the SM if the DataManager fails.
764:  INPUTS:
765:      arg - The buffer containing the SDMRegInfo message.
766:  RETURNS:
767:      void * - Always NULL.
768:  */
769: void* RegInfoHandler(void* arg)
770: {

```

```

771: char* buf = (char*) arg;
772: SDMRegInfo reginfo_msg;
773: SDMConsume consume_msg;
774: long result;
775:
776: // Unmarshal the message
777: if ((result = reginfo_msg.Unmarshal(buf)) < 0)
778: {
779:     if (result == SDM_NO_FURTHER_DATA_PROVIDER)
780:     {
781:         delete [] buf;
782:         return NULL;
783:     }
784:     else
785:     {
786:         printf("Invalid SDMRegInfo message (%ld). \n",result);
787:         delete [] buf;
788:         return NULL;
789:     }
790: }
791: else
792:     MessageReceived(&reginfo_msg);
793: // If the SDMRegInfo message is about the TaskManager's mode notification
794: if (reginfo_msg.id == TM_Mode_ID)
795: {
796:     // Subscribe to this message
797:     consume_msg.source = reginfo_msg.source;
798:     consume_msg.destination.setPort(PORT_SM);
799:     consume_msg.msg_id = reginfo_msg.msg_id;
800:     tm_mode_msg_id = reginfo_msg.msg_id.getInterfaceMessagePair();
801:     tm_mode_source = reginfo_msg.source;
802:     consume_msg.Send();
803:     MessageSent(&consume_msg);
804: }
805:
806: delete [] buf;
807: return NULL;
808: }
809: /*
810: DataHandler handles the receipt of SDMDData message to the SensorManager. The SM
subscribes to the TaskManager's mode change message which

```

```

811:   informs the SM of a DataManager failure. If the SDMData message is from the TaskManager,
the SensorManager acquires the new DM address and
812:   reregisters its xTEDS along with the xTEDS of all of its connected sensors.
813:   INPUTS:
814:       arg - The buffer containing the SDMData message.
815:   RETURNS:
816:       void * - Always NULL.
817: */
818: void* DataHandler(void* arg)
819: {
820:     char* buf = (char*) arg;
821:     SDMData data_msg;
822:     SDMComponent_ID new_dm_address;
823:     unsigned char old_mode;
824:
825:     // Unmarshal the message
826:     if (data_msg.Unmarshal(buf) < 0)
827:     {
828:         printf("Invalid SDMData message. \n");
829:         delete [] buf;
830:         return NULL;
831:     }
832:
833:     // If this is a TaskManager mode change message
834:     if (data_msg.msg_id == tm_mode_msg_id && data_msg.source == tm_mode_source)
835:     {
836:         old_mode = tm_mode;
837:         tm_mode = data_msg.msg[0];
838:
839:         debug_f(1,"TM Mode Change %d. \n",tm_mode);
840:         if (tm_mode == MODE_SOFT_RESET || tm_mode == MODE_HARD_RESET)
841:         {
842:             new_dm_address.Unmarshal(&data_msg.msg[1], 0);
843:
844:             DataManager = new_dm_address;
845:             debug_f(1,"New DM address is 0x%lx:%hd %ld \n",new_dm_address.getAddress(),
new_dm_address.getPort(), new_dm_address.getSensorID());
846:
847:             // Clear the SM's subscription list, it is no longer applicable
848:             pthread_mutex_lock(&subscription_mutex);
849:             sm_subscriptions.ClearAllSubscriptions();

```

```

850:         pthread_mutex_unlock(&subscription_mutex);
851:
852:
853:         // Verify the Data Manager
854:         VerifyDM();
855:
856:         // Resend the SM's xTEDS
857:         SDMxTEDS xteds;
858:         strcpy(xteds.xTEDS, sm_xTEDS);
859:         xteds.source.setPort(PORT_SM);
860:         xteds.source.setAddress(Address_SM);
861:         xteds.source.setSensorID(1);
862:         xteds.Send();
863:         MessageSent(&xteds);
864:
865:         // Re-register all sensors
866:         for (unsigned int i = 0; i < MAX_SENSORS; i++)
867:         {
868:             // Reregister the xTEDS to the DM, of all registered sensors
869:             sensor[i].ReRegister();
870:         }
871:     }
872:     tm_mode = old_mode;
873: }
874:
875: delete [] buf;
876: return NULL;
877: }
878: /*
879: PublishPerformanceCounterMessage publishes the SensorManager's performance counters to all
registered subscribers.
880: INPUTS:
881:     None.
882: RETURNS:
883:     void
884: */
885: void PublishPerformanceCounterMessage(void)
886: {
887:     char msg[16]; // Buffer for performance counters message
888:     // Fill the buffer with current values of the performance counters
889:     pthread_mutex_lock(&perf_counter_mutex);

```



```

890:  PUT_UINT(&msg[0], total_recd);
891:  PUT_UINT(&msg[4], prevsec_recd);
892:  PUT_UINT(&msg[8], total_sent);
893:  PUT_UINT(&msg[12], prevsec_recd);
894:  pthread_mutex_unlock(&perf_counter_mutex);
895:
896:  pthread_mutex_lock(&subscription_mutex);
897:  // Publish the notification
898:  if (sm_subscriptions.Publish(SUB_PERFORMANCE_COUNTERS, msg, 16))
899:  {
900:      // Log message
901:      MessageSent(sm_subscriptions.GetLastPublished());
902:      debug_f(3, "Performance counter notification sent. \n");
903:  }
904:  else
905:  {
906:      debug_f(3, "No performance counter subscribers. \n");
907:  }
908:  pthread_mutex_unlock(&subscription_mutex);
909: }
910:
911: /*
912:  UDPMonitor listens for messages (which should be from the Data Manager) regarding
913:  subscriptions and cancellations for ASIMs and SensorManager
914:  streams and services
915:  INPUTS:
916:      args - Not used, only there to match function signature for thread routines.
917:  RETURNS:
918:      void * - Always NULL
919: */
920: void* UDPMonitor(void* args)
921: {
922:     int result;
923:     char buf[BUFSIZE];
924:     char msg_id;
925:     int sock;
926:     pthread_t HandlerThread;
927:     pthread_attr_t attr;
928:     pthread_attr_init(&attr);
929:     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

```

```

930:
931:  sock = UDPpassive(PORT_SM);
932:  if (sock == IP_SOCKET_INVALID)
933:  {
934:      printf("UDPMonitor::Could not open port %d. \n",PORT_SM);
935:      return NULL;
936:  }
937:  while(1)
938:  {
939:      if(UDPserv_recv(sock, buf, sizeof(buf)) < 0)
940:      {
941:          perror("UDPMonitor: UDPserv_recv error");
942:          UDPclose(sock);
943:          return NULL;
944:      }
945:      msg_id = buf[0];
946:      pthread_mutex_lock(&perf_counter_mutex);
947:      total_recd++;
948:      prevsec_recd++;
949:      pthread_mutex_unlock(&perf_counter_mutex);
950:      switch(msg_id)
951:      {
952:          case SDM_Serreqst:
953:              result = pthread_create(&HandlerThread,&attr,&ServiceHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
954:              if(result < 0)
955:              {
956:                  perror("Could not start Service Request Handler");
957:              }
958:              break;
959:          case SDM_Command:
960:              result = pthread_create(&HandlerThread,&attr,&CommandHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
961:              if(result < 0)
962:              {
963:                  perror("Could not start Command Handler");
964:              }
965:              break;
966:          case SDM_Subreqst:
967:              result = pthread_create(&HandlerThread,&attr,&SubscriptionHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
968:              if(result < 0)

```

```

969:         {
970:             perror("Could not start Subscription Request Handler");
971:         }
972:         break;
973:     case SDM_Deletesub:
974:         result =
pthread_create(&HandlerThread,&attr,&DeleteSubscriptionHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
975:         if(result < 0)
976:         {
977:             perror("Could not start Cancel Request Handler");
978:         }
979:         break;
980:     case SDM_Tat:
981:         result = pthread_create(&HandlerThread,&attr,&TatHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
982:         if(result < 0)
983:         {
984:             perror("Could not start Time At Tone Handler");
985:         }
986:         break;
987:     case SDM_Ready:
988:         result = pthread_create(&HandlerThread,&attr,&ReadyHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
989:         if (result < 0)
990:         {
991:             perror("Could not start Ready Handler");
992:         }
993:         break;
994:     case SDM_RegInfo:
995:         result = pthread_create(&HandlerThread,&attr,&RegInfoHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
996:         if (result < 0)
997:         {
998:             perror("Could not start RegInfo Handler");
999:         }
1000:         break;
1001:     case SDM_Data:
1002:         result = pthread_create(&HandlerThread,&attr,&DataHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1003:         if (result < 0)
1004:         {

```

```

1005:         perror("Could not start Data Handler");
1006:     }
1007:     break;
1008:     case SDM_Error:        //there are currently no error messages handled
1009:         break;
1010:     case SDM_ACK:
1011:         result = pthread_create(&HandlerThread,&attr,&AckHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1012:         if(result < 0)
1013:         {
1014:             perror("Could not start Ack Handler");
1015:         }
1016:     break;
1017:     case SDM_Register:
1018:         result = pthread_create(&HandlerThread,&attr,&RegisterHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1019:         if(result < 0)
1020:         {
1021:             perror("Could not start Register Handler");
1022:         }
1023:     break;
1024:     case SDM_ID:
1025:         result = pthread_create(&HandlerThread,&attr,&IDHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1026:         if(result < 0)
1027:         {
1028:             perror("Could not start ID Handler");
1029:         }
1030:     break;
1031:     default:
1032:         printf("Unexpected Message: %c \n", msg_id);
1033:         SDMAck ackMsg;
1034:         ackMsg.Unmarshal(buf);
1035:         printf("Ack message error code: %i \n", ackMsg.error);
1036:
1037:         break;
1038:     }
1039: }
1040: UDPclose(sock);
1041: return NULL;
1042: }
1043:

```

```

1044: /*
1045:  TCPMonitor listens for TCP Connections used for more robust command delivery.
1046:  INPUTS:
1047:      args - Not used, only there to match function signature for thread routines.
1048:  RETURNS:
1049:      void * - Always NULL
1050: */
1051: void* TCPMonitor(void* args)
1052: {
1053:     int result;
1054:     char buf[BUFSIZE];
1055:     struct sockaddr_in sin;
1056:     int listen_sock;
1057:     int sock;
1058:     int count;
1059:     short length;
1060:     int status;
1061:
1062:     pthread_t HandlerThread;
1063:     pthread_attr_t attr;
1064:
1065:     memset (&sin, 0, sizeof(sin));
1066:     sin.sin_family = AF_INET;
1067:     sin.sin_addr.s_addr = INADDR_ANY;
1068:     sin.sin_port = htons (static_cast<u_int16_t>(PORT_SM));
1069:     listen_sock = TCPpassive(PORT_SM, MAX_TCP_CONNECTIONS);
1070:     if (listen_sock == IP_SOCKET_INVALID) {
1071:         return NULL;
1072:     }
1073:
1074:     pthread_attr_init(&attr);
1075:     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
1076:
1077:     while(1)
1078:     {
1079:         sock = TCPServ_accept(listen_sock);
1080:         if (sock < 0) // cleanup issue 31
1081:         {
1082:             printf("Error: TCPMonitor could not accept socket connection. \n");
1083:             return NULL;
1084:         }

```

```

1085:     status = TCPRecv(sock,buf,BUFSIZE); //get message
1086: if (status < 0)
1087: {
1088:     printf("Error: TCPMonitor: recv error \n");
1089:     return NULL;
1090: }
1091: count = status;
1092:     while(count < HEADER_SIZE)
1093: {
1094:     status = TCPRecv(sock,&buf[status],BUFSIZE-status);
1095: if (status < 0)
1096: {
1097:     printf("Error: TCPMonitor: recv error \n");
1098:     return NULL;
1099: }
1100: count += status;
1101: }
1102:     length = GET_SHORT(&buf[9]);
1103:     while(length > status - HEADER_SIZE)
1104: {
1105:     status = TCPRecv(sock,&buf[status],BUFSIZE-status);
1106: if (status < 0) // cleanup issue 32
1107: {
1108:     return NULL;
1109: }
1110: count += status;
1111: }
1112: if(length != status - HEADER_SIZE)
1113:     printf("Error: More bytes recv'd than expected! \n");
1114:
1115: pthread_mutex_lock(&perf_counter_mutex);
1116: total_recd++;
1117: prevsec_recd++;
1118: pthread_mutex_unlock(&perf_counter_mutex);
1119:
1120: switch(buf[0])
1121: {
1122: case SDM_Serreqst:
1123:     result = pthread_create(&HandlerThread,&attr,&ServiceHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1124:     if(result < 0)

```

```

1125:     {
1126:         perror("Could not start Service Request Handler");
1127:     }
1128:     break;
1129:     case SDM_Command:
1130:         result = pthread_create(&HandlerThread,&attr,&CommandHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1131:         if(result < 0)
1132:         {
1133:             perror("Could not start Command Handler");
1134:         }
1135:         break;
1136:     case SDM_Subreqst:
1137:         result = pthread_create(&HandlerThread,&attr,&SubscriptionHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1138:         if(result < 0)
1139:         {
1140:             perror("Could not start Subscription Request Handler");
1141:         }
1142:         break;
1143:     case SDM_Deletesub:
1144:         result = pthread_create(&HandlerThread,&attr,&DeleteSubscriptionHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1145:         if(result < 0)
1146:         {
1147:             perror("Could not start Cancel Request Handler");
1148:         }
1149:         break;
1150:     case SDM_Tat:
1151:         result = pthread_create(&HandlerThread,&attr,&TatHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1152:         if(result < 0)
1153:         {
1154:             perror("Could not start Time At Tone Handler");
1155:         }
1156:         break;
1157:     case SDM_Ready:
1158:         result = pthread_create(&HandlerThread,&attr,&ReadyHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1159:         if (result < 0)
1160:         {

```

```

1161:         perror("Could not start Ready Handler");
1162:     }
1163:     break;
1164:     case SDM_RegInfo:
1165:         result = pthread_create(&HandlerThread,&attr,&RegInfoHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1166:         if (result < 0)
1167:         {
1168:             perror("Could not start RegInfo Handler");
1169:         }
1170:         break;
1171:     case SDM_Data:
1172:         result = pthread_create(&HandlerThread,&attr,&DataHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1173:         if (result < 0)
1174:         {
1175:             perror("Could not start Data Handler");
1176:         }
1177:         break;
1178:     case SDM_Error:        //there are currently no error messages handled
1179:         break;
1180:     case SDM_ACK:
1181:         result = pthread_create(&HandlerThread,&attr,&AckHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1182:         if(result < 0)
1183:         {
1184:             perror("Could not start Ack Handler");
1185:         }
1186:         break;
1187:     case SDM_Register:
1188:         result = pthread_create(&HandlerThread,&attr,&RegisterHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1189:         if(result < 0)
1190:         {
1191:             perror("Could not start Register Handler");
1192:         }
1193:         break;
1194:     case SDM_ID:
1195:         result = pthread_create(&HandlerThread,&attr,&IDHandler,memcpy(new
char[BUFSIZE],buf,BUFSIZE));
1196:         if(result < 0)
1197:         {

```



```

1198:         perror("Could not start ID Handler");
1199:     }
1200:     break;
1201: }
1202:     TCPclose(sock);
1203: }
1204: return NULL;
1205: }
1206: /*
1207:  VerifyDM is used to verify the existence of the DataManager when the SM is started. The
1208:  existence of the DM is necessary for any SDM
1209:  component to operate.
1210:  INPUTS:
1211:  None.
1212:  RETURNS:
1213:  void
1214:  */
1215: void VerifyDM(void)
1216: {
1217:     bool msg_rcvd = false;
1218:     SDMReady msg;
1219:     SDMReady msg_in;
1220:     msg.destination.setPort(PORT_SM);
1221:     msg.destination.setAddress(Address_SM);
1222:     msg.source.setPort(PORT_SM);
1223:     printf("Searching for DM ..");
1224:     fflush(NULL);
1225:     do {
1226:         pthread_mutex_lock(&perf_counter_mutex);
1227:         total_sent++;
1228:         prevsec_sent++;
1229:         pthread_mutex_unlock(&perf_counter_mutex);
1230:         msg.SendBCast(DataManager.getAddress(), PORT_DM);
1231:         putchar('.');
1232:         fflush(NULL);
1233:         usleep(250000);
1234:         pthread_mutex_lock(&dm_found_mutex);
1235:         msg_rcvd = dm_found;
1236:         pthread_mutex_unlock(&dm_found_mutex);
1237:     } while (!msg_rcvd);
1238:     printf("DM found \n");

```

```

1238: }
1239: /*
1240:  GetNodeAddress sets the IP address of the SensorManager in the global variable Address_SM.
1241:  Returns: int - Zero on success, -1 on failure.
1242: */
1243: #ifdef WIN32
1244: int GetNodeAddress()
1245: {
1246:  hostent * localHost;
1247:  char * IPAddr;
1248:  char hostname[64];
1249:
1250:  if (gethostname(hostname,sizeof(hostname)) < 0)
1251:      return -1;
1252:  if ((localHost = gethostbyname(hostname)) == NULL)
1253:      return -1;
1254:  IPAddr = inet_ntoa(*(struct in_addr *)localHost->h_addr_list[0]);
1255:  Address_SM = inet_addr(IPAddr);
1256:  debug_f(3,"Sensor Manager address is 0x%x\n",Address_SM);
1257:  return 0;
1258: }
1259: #else
1260: int GetNodeAddress()
1261: {
1262:  struct ifreq ifr;
1263:  struct sockaddr_in *sin = (struct sockaddr_in *)&ifr.ifr_addr;
1264:  int sockfd;
1265:
1266:  bzero(&ifr, sizeof(ifr));
1267:
1268:  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
1269:  {
1270:      perror("Error GetNodeAddres(), socket(): ");
1271:      return(-1);
1272:  }
1273:
1274:  strcpy(ifr.ifr_name, "eth0");
1275:  sin->sin_family = AF_INET;
1276:
1277:  if(ioctl(sockfd, SIOCGIFADDR, &ifr) == 0)
1278:  {

```

```

1279:     Address_SM = inet_addr(inet_ntoa(sin->sin_addr));
1280:     debug_f(3,"SM Address is %ld \n",Address_SM);
1281: }
1282: else
1283:     return -1;
1284: return 0;
1285: }
1286: #endif
1287: /*
1288:  MessageSent is called whenever an SDM* message is sent within the SensorManager. This
1289:  function increments the performance counters and
1290:  logs the message if the message logger is set.
1291:  INPUTS:
1292:      msg - A pointer to the SDM* message sent.
1293:  RETURNS:
1294:      void
1295:  */
1296: void MessageSent(SDMmessage *msg)
1297: {
1298:     pthread_mutex_lock(&perf_counter_mutex);
1299:     total_sent++;
1300:     prevsec_sent++;
1301:     pthread_mutex_unlock(&perf_counter_mutex);
1302:     pthread_mutex_lock(&log_service_mutex);
1303:     if (!log_service.IsEmpty())
1304:         log_service.MessageSent(msg);
1305:     pthread_mutex_unlock(&log_service_mutex);
1306: }
1307: /*
1308:  MessageReceived is called whenever an SDM* message is received within the SensorManager.
1309:  This function logs the message if the log
1310:  service is set.
1311:  INPUTS:
1312:      msg - A pointer to the SDM* message received.
1313:  RETURNS:
1314:      void
1315:  */
1316: void MessageReceived(SDMmessage *msg)
1317: {
1318:     pthread_mutex_lock(&log_service_mutex);
1319:     if (!log_service.IsEmpty())

```

```
1318:     log_service.MessageReceived(msg);
1319: pthread_mutex_unlock(&log_service_mutex);
1320: }
```

File: sdm/sm/Makefile

```
1: include ../Makefile.common
2: include ../$(MAKEFILE_DEFS)
3:
4: .PHONY:    all clean distclean
5:
6: all: sm sm_process
7:
8: sm_process: sm_process.o sensor.o SensorRecord.o SensorMonitor.o
9:  $(CXX) $(CXXFLAGS) -L../common -static -o $@ $^ $(BOOSTFLAGS) -lSDM -lpthread
10:
11: sensor.o: sensor.cpp sensor.h
12:  $(CXX) $(CXXFLAGS) -c $<
13:
14: sm_process.o:  sm.cpp sm.h
15:  $(CXX) $(CXXFLAGS) -c -o $@ $<
16:
17: SensorRecord.o: SensorRecord.cpp SensorRecord.h
18:  $(CXX) $(CXXFLAGS) -c -o $@ $<
19:
20: SensorMonitor.o: SensorMonitor.cpp SensorMonitor.h SensorRecord.o
21:  $(CXX) $(CXXFLAGS) -c -o $@ $<
22:
23: sm: sm_monitor.cpp
24:  $(CXX) $(CXXFLAGS) -L../common -static -o $@ $< $(BOOSTFLAGS) -lSDM -lpthread
25:
26: clean:
27:  rm -f *.o *~ *.out SDMMessages*
28:
29: distclean: clean
30:  rm -f sm sm_process
```

File: sdm/sm/SensorRecord.cpp

```
1: #include "SensorRecord.h"
2:
3: SensorRecord::SensorRecord() : m_Registered(false), m_SecondsStart(SECONDS_INIT_VAL),
ackReceived(false), regConfirmed(false), helloSentTime(SECONDS_INIT_VAL),
xTEDSSentTime(SECONDS_INIT_VAL)
4: {}
5:
6: SensorRecord::~SensorRecord()
7: {}
8:
9: bool SensorRecord::HasExpired(unsigned int CurSeconds)
10: {
11: if (CurSeconds == SECONDS_INIT_VAL)
12:     return false;
13:
14: return CurSeconds > m_SecondsStart ? true : false;
15: }
16:
17: bool SensorRecord::CheckAckTimeout(unsigned int curSeconds)
18: {
19: if(curSeconds == SECONDS_INIT_VAL)
20: {
21:     return false;
22: }
23: else if(curSeconds > helloSentTime + SENSOR_ACK_TIMEOUT)
24: {
25:     return true;
26: }
27: else
28: {
29:     return false;
30: }
31: }
32:
33:
34: bool SensorRecord::CheckIDTimeout(unsigned int curSeconds)
35: {
36: if(curSeconds == SECONDS_INIT_VAL)
37: {
```

```
38:     return false;
39: }
40: else if(curSeconds > xTEDSSentTime + SENSOR_ID_TIMEOUT)
41: {
42:     return true;
43: }
44: else
45: {
46:     return false;
47: }
48: }
49:
```

File: sdm/sm/sensor.h

```
1: #ifndef _SENSOR_H_
2: #define _SENSOR_H_
3:
4: #define SM_OFFSET    2        // offset of sensor_id from asim number
5:
6: #include <pthread.h>
7: #include "../common/SubscriptionManager/SubscriptionManager.h"
8: #ifdef WIN32
9: #include "../common/asim/asim_win32.h"
10: #else
11: #include "../common/asim/ASIM.h"
12: #endif
13: #include "../common/message/SDMSubreqst.h"
14: #include "../common/message/SDMDeletesub.h"
15: #include "../common/message/SDMCommand.h"
16: #include "../common/message/SDMSerreqst.h"
17: #include "../common/message/SDMComponent_ID.h"
18:
19: extern void MessageSent(SDMmessage *msg);
20:
21: class SensorMonitor;
22: class Sensor
23: {
24: public:
25:   Sensor(void);
26:   Sensor(long);
27:   Sensor(const Sensor&);
28:   ~Sensor();
29:
30:   // Public operations
31:   char Read(unsigned short& length, unsigned char* buf, int bufsize);
32:   void Subscribe(SDMSubreqst&);
33:   void CancelSubscription(SDMDeletesub&);
34:   void Service(SDMSerreqst&);
35:   void Command(SDMCommand&);
36:   void TimeAtTone(unsigned long seconds,unsigned long useconds);
37:   void SetDebug(int);
38:   void ReRegister();
39:   Sensor& operator=(const Sensor&);
```



```

40: bool CheckConnection();
41: pthread_t* StartUSBMonitor(long, SensorMonitor* Monitor);
42: void CancelSensor(bool Disconnected);
43:
44: // Public data members
45: ASIM device;
46: SDMComponent_ID sensor_id;
47: char *device_xTEDS;
48:
49: private:
50: // Private methods
51: void Cancel();
52: bool Register(int sensorIndex);
53: static void* USBMonitor(void*);
54: bool Open(long);
55:
56: // Private data members
57: SubscriptionManager subscriptions;
58: char device_name[15];
59: pthread_mutex_t sensor_mutex;      //Locks ASIM writes and read/write from the subscriptions
member
60: bool connected;
61: bool m_Ready;
62: int debug_level;
63: pthread_t m_USBMonitorThread;
64: SensorMonitor* m_Monitor;
65: };
66:
67: class USBMonitorThreadArgs
68: {
69: public:
70: USBMonitorThreadArgs(Sensor* Sens, SensorMonitor* Mon, int ID) : SensorInstance(NULL),
MonitorInstance(NULL), SensorID(-1)
71: { SensorInstance = Sens; MonitorInstance = Mon; SensorID = ID; }
72: Sensor* SensorInstance;
73: SensorMonitor* MonitorInstance;
74: int SensorID;
75: };
76:
77: #endif

```

File: sdm/sm/sm_monitor.cpp

```
1: /**
2:  * Sensor Manager monitor - main program unit
3:  *
4:  * The Sensor Manager monitor starts the Sensor Manager (SM) process, and
5:  * ensures that it is up and responding. The monitor periodically sends a
6:  * heartbeat message to the SM. If this monitor detects no or improper response
7:  * from the SM, the SM process will be restarted. This provides a level of
8:  * robustness for the SM.
9:  *
10:  * See sm.cpp for reference information on the command line arguments, which
11:  * are passed through from this Sensor Manager monitor to the Sensor Manager
12:  * process.
13:  */
14:
15: #include <stdio.h>
16: #include <unistd.h>
17: #include <sys/types.h>
18: #include <signal.h>
19: #include <stdlib.h>
20: #include <arpa/inet.h>
21: #include <netinet/in.h>
22: #include <sys/socket.h>
23: #include <sys/wait.h>
24: #include <string.h>
25: #include "../common/message/SDMReady.h"
26: #include "../common/MessageManager/MessageManager.h"
27: #include "../common/TCPcom.h"
28:
29: #define NUM_HEARTBEAT_TRIES    2    //Number of times to miss a response before restarting
the SM
30:
31: pid_t StartSM(char **); // Starts the Sensor Manager process
32: void SigIntHandler(int); // The non-default signal handler for SIGINT
33:
34: pid_t sm_pid = -1;      //Process id number of the Sensor Manager
35:
36:
37: /**
38:  * main - entry point for SM monitor process
```

```

39: *
40: * @param argc - the argument count
41: * @param argv - the command arguments
42: *
43: * @return >= 0 for no error, < 0 for error
44: */
45: int main(int argc, char ** argv)
46: {
47:     int result = -1;    // set with Termination status for child process
48:     int miss_count = 0; // Count of number of heartbeat messages missed
49:     int num_toget = 0; // The number of messages to receive per
50:                          // heartbeat
51:     char buf[BUFSIZE]; // Buffer for message receipt
52:     SDMReady heartbeat; // Heart-beat message instance
53:     MessageManager mm; // Message manager helper instance
54:     SDMComponent_ID sm; // ID information on Sensor Manager process
55:
56:     const size_t max_proc_size = 128; // Limit the process name size to
57:                                     // reasonable length
58:     size_t prog_size = 0; // The size to build the process name
59:     size_t argv0_size = 0; // The length of argv[0]
60:     size_t suffix_size = 0; // The length of the suffix to append to argv[0]
61:     char *proc_name = 0; // The buffer pointer for building process-name
62:     const char* proc_suffix = "_process"; // Suffix to add to the incoming
63:                                     // process name
64:
65:     const char* localhostIp = "127.0.0.1"; // The standard local host IP address
66:
67:     //////////////////////////////////////
68:     // Set up the command line for the child process
69:     // Prepares a new argv[0] for SM with name of <this_process>_process
70:     //////////////////////////////////////
71:     argv0_size = strlen(argv[0]);
72:     if (argv0_size > max_proc_size)
73:     {
74:         printf(" Monitor: Command path too long for SM. \n");
75:         return(-1);
76:     }
77:
78:     suffix_size = strlen(proc_suffix);
79:

```

```

80: // Allocate for full size, including terminator and for the '_process' suffix,
81: // but limit the length to reasonable length
82: if(argv0_size + suffix_size + 1 < max_proc_size)
83: {
84:     prog_size = argv0_size + suffix_size + 1;
85: }
86: else
87: {
88:     prog_size = max_proc_size;
89: }
90:
91: try
92: {
93:     proc_name = new char[prog_size];
94: }
95: catch(...)
96: {
97:     printf(" Monitor: Error allocating %u bytes for command name. \n",
98:     prog_size);
99:     return(-1);
100: }
101:
102: // set a size to ensure to leave enough space in buffer for suffix to be
103: // appended
104: size_t max_argv_chars;
105: if(max_proc_size - (suffix_size + 1) < argv0_size)
106: {
107:     max_argv_chars = max_proc_size - (suffix_size + 1);
108: }
109: else
110: {
111:     max_argv_chars = argv0_size;
112: }
113: strncpy(proc_name, argv[0], max_argv_chars);
114: if ( max_argv_chars > 0 )
115:     proc_name[max_argv_chars] = '\0'; // ensure termination
116: strcat(proc_name, proc_suffix); // append the suffix
117:
118: //////////////////////////////////////
119: // Process the command line arguments...
120: // A copy of the argv's are prepared to pass along to the SM instance. The

```

```

121: // exception is that argv[0] is prepared as per above with a suffix.
122: //////////////////////////////////////
123: char **nargv = new char*[argc+1];
124: nargv[0] = proc_name;
125: for (int i = 1; i < argc; ++i)
126:     nargv[i] = argv[i];
127: nargv[argc] = NULL;
128:
129:
130: //////////////////////////////////////
131: // Prepare for the heartbeat messaging...
132: //////////////////////////////////////
133:
134: //Set monitor process address
135: heartbeat.source.setSensorID(0);
136: heartbeat.source.setAddress(inet_addr(localhostIp));
137: heartbeat.source.setPort(PORT_SM_MONITOR);
138:
139: //Set SM address
140: sm.setAddress(inet_addr(localhostIp));
141: sm.setPort(PORT_SM);
142: sm.setSensorID(0);
143:
144: // Setup a signal handler for SIGINT, the user-keyboard interrupt signal
145: signal(SIGINT, SigIntHandler);
146:
147:
148: //////////////////////////////////////
149: // Start the SM in a seperate process
150: //////////////////////////////////////
151:
152: sm_pid = StartSM(nargv);
153:
154: //////////////////////////////////////
155: //
156: //////////////////////////////////////
157:
158: mm.Async_Init_Both(PORT_SM+1);
159:
160: //////////////////////////////////////
161: // Start the heartbeat monitoring...

```

```

162: // If SM quits, or no messages received in allowed time frame, then restart
163: // the SM
164: //////////////////////////////////////
165:
166: //If fork/exec was successful
167: if (sm_pid > 0)
168: {
169:     //Allow the SM to get started up
170:     sleep(HEARTBEAT_INTERVAL);
171:     while (1)
172:     {
173:
174:         //Send heartbeats via UDP and TCP
175:         heartbeat.SendTo(sm);
176:         heartbeat.Sendtcp(sm);
177:         sleep(HEARTBEAT_INTERVAL);
178:
179:         //If Sensor Manager quit
180:         if (waitpid(-1,&result,WNOHANG) == sm_pid)
181:         {
182:             printf(" Monitor: SM failed, restarting... \n");
183:             sm_pid = StartSM(nargv);
184:             if (sm_pid > 0)
185:             {
186:                 sleep(HEARTBEAT_INTERVAL);
187:                 continue;
188:             }
189:             else
190:             {
191:                 printf(" Monitor: Could not restart the SM. \n");
192:                 return -1;
193:             }
194:         }
195:         else if (mm.IsReady())
196:         {
197:             //Should respond with two messages
198:             num_toget = 2;
199:
200:             while (mm.IsReady())
201:             {
202:                 num_toget--;

```

```

203:    mm.GetMessage(buf);
204: }
205: if (num_toget > 0)
206:     miss_count++;
207: else
208:     miss_count = 0;
209: }
210: else
211: {
212: //If no messages were received
213: miss_count++;
214: if (miss_count == NUM_HEARTBEAT_TRIES)
215: {
216:     printf(" Monitor: SM unresponsive, restarting... \n");
217:
218:     if (kill (sm_pid, SIGKILL) == 0)
219:         wait(&result);
220:     else
221:         waitpid(-1, &result, WNOHANG);
222:
223:     sm_pid = StartSM(nargv);
224:     if (sm_pid > 0)
225:     {
226:         sleep(HEARTBEAT_INTERVAL);
227:         continue;
228:     }
229:     else
230:     {
231:         printf(" Monitor: Could not restart the SM. \n");
232:         return -1;
233:     }
234: }
235: }
236: }
237: }
238: else
239: {
240:     printf(" Monitor: Error starting the SM (%d). \n", sm_pid);
241:     // Cleanup allocations
242:     delete [] proc_name;
243:     return -1;

```

```

244: }
245:
246: // Cleanup allocations
247: delete [] proc_name;
248:
249: return 0;
250: }
251:
252: /**
253:  * Start the Sensor Manager as a new process
254:  *
255:  * Post fork call, if this is the child process, the call to execvp(...) will
256:  * overlay this process, and hence replace exit from this function.
257:  *
258:  * @param argv - the argument list for sensor manager.
259:  * @return - the process ID of the Sensor Manager child process
260:  */
261: int StartSM(char ** argv)
262: {
263:     int pid;
264:
265:     pid = fork();
266:     //Child Process
267:     if (pid == 0)
268:     {
269:         //Start the Sensor Manager
270:         if (execvp(argv[0],argv) < 0)
271:         {
272:             printf(" Monitor: Error exec'ing the SM. \n");
273:             exit(-1);
274:         }
275:         //Here to make the compiler happy, never runs however
276:         return 0;
277:     }
278:     //Parent Process
279:     else
280:         return pid;
281: }
282:
283: /**
284:  * Provides a handler for the SIGINT (Terminal Interrupt) signal

```



```
285: *
286: * @param sig_num - the argument list for sensor manager.
287: *
288: * @return - the process ID of the Sensor Manager child process
289: *
290: */
291: void SigIntHandler(int sig_num)
292: {
293:     int result = 0;
294:     if (kill (SIGINT, sm_pid) == 0)
295:         wait (&result);
296:     else
297:         waitpid(-1,&result,WNOHANG);
298:
299:     exit(EXIT_SUCCESS);
300: }
301:
```

File: sdm/sm/SensorMonitor.h

```
1: #ifndef _SENSOR_MONITOR_H_
2: #define _SENSOR_MONITOR_H_
3:
4: #include <pthread.h>
5: #include "SensorRecord.h"
6: #include "sensor.h"
7: #include "../common/Time/SDMTime.h"
8:
9: class SensorMonitor
10: {
11: public:
12:     SensorMonitor();
13:     ~SensorMonitor();
14:     SensorMonitor(const SensorMonitor&);
15:     SensorMonitor& operator = (const SensorMonitor&);
16:
17:     bool InitializeSensors(unsigned int NumSensors, bool Prompt, int DebugLevel);
18:     bool SensorRegistering(int SensorNum);
19:     bool SensorRegistered(int SensorNum);
20:     bool SensorAcked(int SensorNum);
21:     bool SensorRegConfirmed(int SensorNum);
22:     unsigned int SetHelloSendTime(int SensorNum);
23:     unsigned int SetxTEDSSendTime(int SensorNum);
24:     bool SendxTEDS(int SensorNum);
25:     Sensor& operator[] (int Index);
26: private:
27:     static void* SensorMonitorFunc(void*);
28:
29:     pthread_t m_SensorMonitorThread;
30:     int m_NumSensors;
31:     Sensor* m_Sensors;
32:
33:     pthread_mutex_t m_SensorDataMutex;
34:     SensorRecord* m_SensorData;
35:
36: };
37:
38:
39: #endif
```

File: sdm/sm/sm_monitor_win32.cpp

```
1: //*****
2: //sm_monitor
3: //
4: //The sm_monitor adds robustness to the SensorManager. The monitor periodically sends
5: //heartbeat messages to the SensorManager module residing on the same node. If for
6: //some reason, the SM fails to respond, the monitor attempts to restart it.
7: //*****
8:
9: #include <stdio.h>
10: #include <unistd.h>
11: #include <sys/types.h>
12: #include <signal.h>
13: #include <stdlib.h>
14: #include <arpa/inet.h>
15: #include <netinet/in.h>
16: #include <sys/socket.h>
17: #include <sys/wait.h>
18: #include <string.h>
19: #include "../common/message/SDMReady.h"
20: #include "../common/MessageManager/MessageManager.h"
21: #include "../common/TCPcom.h"
22: #include <windows.h>
23: #include <process.h>
24:
25: #define NUM_HEARTBEAT_TRIES 2 /*Number of times to miss a response before restarting
the SM*/
26:
27: HANDLE StartSM(char **);
28: void SigIntHandler(int);
29:
30: HANDLE sm_handle = NULL; /* Process handle of the SensorManager */
31:
32: int main(int argc, char ** argv)
33: {
34: unsigned long result = 0;
35: int length = 0;
36: int miss_count = 0;
37: int num_toget = 0;
38: char buf[BUFSIZE];
```

```

39: SDMReady heartbeat;
40: MessageManager mm;
41: SDMComponent_ID sm;
42:
43: //Set monitor process address
44: heartbeat.source.setSensorID(0);
45: heartbeat.source.setAddress(inet_addr("127.0.0.1"));
46: heartbeat.source.setPort(PORT_SM_MONITOR);
47:
48: //Set SM address
49: sm.setAddress(inet_addr("127.0.0.1"));
50: sm.setPort(PORT_SM);
51: sm.setSensorID(0);
52:
53: signal(SIGINT, SigIntHandler);
54: sm_handle = StartSM(argv);
55: mm.Async_Init_Both(PORT_SM+1);
56:
57: /* If spawn was successful */
58: if (sm_handle != NULL)
59: {
60:     /* Allow the SM to get started up */
61:     sleep(HEARTBEAT_INTERVAL);
62:     while (1)
63:     {
64:         //Send heartbeats via UDP
65:         heartbeat.SendTo(sm);
66:         length = heartbeat.Marshal(buf);
67:         sleep(HEARTBEAT_INTERVAL);
68:
69:         //If Sensor Manager quit
70:         if (!GetExitCodeProcess(sm_handle,&result))
71:             printf(" Monitor: Invalid handle for retrieving exit code. \n");
72:         /* If the SensorManager has quit for some reason */
73:         if (result != STILL_ACTIVE)
74:         {
75:             printf(" Monitor: SM failed, restarting... \n");
76:             sm_handle = StartSM(argv);
77:             if (sm_handle != NULL)
78:             {
79:                 /* Allow the SM to restart */

```

```

80:         sleep(HEARTBEAT_INTERVAL);
81:         /* Start over at while */
82:         continue;
83:     }
84:     /* Error re-starting SM */
85:     else
86:     {
87:         printf(" Monitor: Could not restart the SM. \n");
88:         return -1;
89:     }
90: }
91: /* The SM is still running, check for responses to heartbeat messages */
92: else if (mm.IsReady())
93: {
94:     //Should respond with one message
95:     num_toget = 1;
96:
97:     while (mm.IsReady())
98:     {
99:         num_toget--;
100:        /* Message is discarded, we only care that one was received */
101:        mm.GetMsg(buf);
102:    }
103:    if (num_toget > 0)
104:        miss_count++;
105:    else
106:        miss_count = 0;
107: }
108: /* The SM is still running, but no responses were received */
109: else
110: {
111:     //If no messages were received
112:     miss_count++;
113:     if (miss_count == NUM_HEARTBEAT_TRIES)
114:     {
115:         printf(" Monitor: SM unresponsive, restarting... \n");
116:         /* If the Terminate fails and the SM is already dead */
117:         if (!TerminateProcess(sm_handle,0))
118:         {
119:             /* Get its return value */
120:             GetExitCodeProcess(sm_handle,&result);

```

```

121:         }
122:         /* Restart the SM */
123:         sm_handle = StartSM(argv);
124:         /* Success restarting */
125:         if (sm_handle != NULL)
126:         {
127:             /* Allow SM to get started */
128:             sleep(HEARTBEAT_INTERVAL);
129:             /* Start over at while loop */
130:             continue;
131:         }
132:         /* Error restarting */
133:         else
134:         {
135:             printf(" Monitor: Could not restart the SM. \n");
136:             return -1;
137:         }
138:     }
139: }
140: }
141: }
142: else
143: {
144:     printf(" Monitor: Error starting the SM. \n");
145:     return -1;
146: }
147: return 0;
148: }
149:
150: /*

```

151: StartSM starts the SensorManager module of the SDM from the monitor process. All command line arguments are passed from the

152: monitor process. This function assumes that both the SensorManager and the monitor binaries reside in the same directory.

153: INPUTS:

154: argv - The command line arguments passed to the SensorManager.

155: RETURNS:

156: HANDLE - The handle to the spawned SensorManager, or NULL if an error has occurred.

157: */

158: HANDLE StartSM(char ** argv)

159: {

```

160:  int p_handle;                                /* Handle to the spawned process (SensorManager)
*/
161:  char proc_name[512];                          /* Buffer for command to execute */
162:  memset(proc_name,' \0',sizeof(proc_name));/* Clear the char buffer */
163:
164:  /* Assure that arguments have been passed */
165:  if (argv)
166:  {
167:      /*argv[0] contains the binary name from the command line */
168:      /*If invoked on command line without .exe */
169:      if (strstr(argv[0],".exe") == NULL)
170:          strncpy(proc_name,argv[0],strlen(argv[0])-8);  /* Copying everything but _monitor */
171:      /*If invoked with .exe */
172:      else
173:          strncpy(proc_name,argv[0],strlen(argv[0])-12); /* Copying everything but _monitor.exe
*/
174:      strcat(proc_name, ".exe");                  /* String is now ...sm.exe */
175:  }
176:  /* If no command line arguments, error */
177:  else
178:      return NULL;
179:  /* Start the SensorManager */
180:  p_handle = _spawnv(_P_NOWAIT,proc_name,argv);
181:  /* If error */
182:  if (p_handle == -1)
183:  {
184:      perror("");
185:      return NULL;
186:  }
187:  /* Return the handle */
188:  else
189:      return (HANDLE)p_handle;
190: }
191: /*
192:  SigIntHandler handles a CTRL+C (SIGINT) signal. This function will kill the SensorManager
and the monitor process.
193:  INPUTS:
194:      sig_num - The signal sent.
195:  RETURNS:
196:      void - None.
197:  */
198: void SigIntHandler(int sig_num)

```

```
199: {
200:   unsigned long result = 0;
201:   if (!sm_handle)
202:     exit(EXIT_SUCCESS);
203:   sleep(1);
204:   if (!TerminateProcess(sm_handle,0))
205:     GetExitCodeProcess(sm_handle,&result);
206:   exit(EXIT_SUCCESS);
207: }
208:
```


Listing from directory: sdm/Spa1Manager

File: sdm/Spa1Manager/Spa1Translator.h

```
1: #ifndef __SPA1_TRANSLATOR_H__
2: #define __SPA1_TRANSLATOR_H__
3:
4: #include "Spa1Asim.h"
5: #include "../common/Debug.h"
6: #include "../common/message/SDMSerreqst.h"
7: #include "../common/message/SDMCommand.h"
8: #include "../common/message/SDMSubreqst.h"
9: #include "../common/message/SDMDeletesub.h"
10:
11: #define SECONDARY_HEADER_LEN 2
12:
13: #define SPA1_ASIM_COMMAND      'V'
14: #define SPA1_ASIM_DATAREQ      'M'
15: #define SPA1_ASIM_CANCEL_DATA 'C'
16: #define SPA1_ASIM_TAT          'O'
17:
18: /**
19:  * @class Spa1Translator
20:  *
21:  * @brief Defines the Spa1Translator object for use by the SPA-1 Sensor Manager
22:  *
23:  * The Spa1Translator takes a message in SDM format and translates that message into
24:  * its equivalent SPA-1 protocol form. It is also used for packaging SPA-1 messages
25:  * to cancel a data stream.
26:  *
27:  * @author Bryan Hansen
28:  * @date 1/13/2010
29:  */
30: class Spa1Translator
31: {
32: public:
33:     Spa1Translator();
34:
35:     int translateToSpa1(unsigned char* inBuf, unsigned char* outBuf, size_t outBufSize);
36:     int packageCancelData(unsigned char intId, unsigned char msgId, unsigned char* outBuf, size_t
outBufSize);
37:
```

```
38: private:
39: int translateSerreqst(unsigned char* inBuf, unsigned char* outBuf, size_t outBufSize);
40: int translateCommand(unsigned char* inBuf, unsigned char* outBuf, size_t outBufSize);
41: int translateSubreqst(unsigned char* inBuf, unsigned char* outBuf, size_t outBufSize);
42: int translateTat(unsigned char* inBuf, unsigned char* outBuf, size_t outBufSize);
43:
44: };
45:
46: #endif
```

File: sdm/Spa1Manager/Spa1AsimTable.h

```
1: #ifndef __SPA1_ASIM_TABLE_H__
2: #define __SPA1_ASIM_TABLE_H__
3:
4: #include <stdio.h>
5: #include <unistd.h>
6: #include <string.h>
7: #include <pthread.h>
8: #include "Spa1Asim.h"
9: #include "Spa1Translator.h"
10: #include "../common/Debug.h"
11: #include "../common/message/SDMCancelxTEDS.h"
12: #include "../common/message/SDMData.h"
13:
14: /**
15:  * @class Spa1AsimTable
16:  *
17:  * @brief Defines the Spa1AsimTable object for use by the SPA-1 Manager
18:  *
19:  * @author Bryan Hansen
20:  * @date 01/13/2010
21:  */
22: class Spa1AsimTable
23: {
24: public:
25:     Spa1AsimTable();
26:     bool checkAddressUsed(unsigned char address);
27:     void addAsim(Spa1Asim* asim);
28:     void removeAsim(unsigned char address);
29:     unsigned char getNextAddress();
30:     Spa1Asim* getAsimBySensorId(unsigned long sensorId);
31:     Spa1Asim* getAsimByGuid(int guid);
32:     Spa1Asim* getAsimBySensorIndex(unsigned short sensorIndex);
33:     void sendToAsims();
34:     void recvFromAsims();
35:     void checkMsgsFromAsim();
36:     void cancelSdmRegistrations();
37:     int getAsimCount();
38:     void printAsimData();
39:     void printLinkedList();
```

```
40:
41: private:
42: ///Next sensor index to assign when a ASIM is discovered
43: unsigned short nextSensorIndex;
44:
45: ///Pointer to the head of the table (linked list)
46: Spa1Asim* head;
47:
48: //Pointer to the tail of the table (linked list)
49: Spa1Asim* tail;
50:
51: ///An array used to specify used and available addresses in the I2C bus
52: bool addressSpace[128];
53:
54: ///The number of ASIMs currently registered
55: int asimCount;
56:
57: ///Mutex which must be held when accessing the Spa1AsimTable
58: pthread_mutex_t asimTableMutex;
59: };
60:
61: #endif
```

File: sdm/Spa1Manager/Spa1Msg.cpp

```
1: #include "Spa1Msg.h"
2:
3: extern int debug;
4:
5: /**
6:  * Constructor that creates a Spa1Msg object with a msg buffer of the size
7:  * passed in as a parameter
8:  *
9:  * @param size The size of a msg, must be less than or equal to 256 bytes
10: */
11: Spa1Msg::Spa1Msg(int size) : msgSize (0)
12: {
13:     if (size > MAX_SPA1_MSG_SIZE)
14:     {
15:         printf("Spa1Msg: Error, requested msg larger than the maximum SPA-1 msg size \n");
16:         msgSize = MAX_SPA1_MSG_SIZE;
17:     }
18:     else
19:     {
20:         msgSize = size;
21:     }
22: }
23:
24: /**
25:  * Destructor
26:  */
27: Spa1Msg::~Spa1Msg()
28: {
29: }
30:
31: /**
32:  * Returns the size of the Spa1Msg objects msg buffer
33:  *
34:  * @return An int containing the size of the msg buffer
35:  */
36: int Spa1Msg::getMsgSize()
37: {
38:     return msgSize;
39: }
```

```

40:
41: /**
42:  * Sets the message size of a Spa1Msg
43:  *
44:  * @param size Specifies the size of the buffer to allocate, should be
45:  *          between 0 and 256 bytes.
46:  * @return An int, 0 indicating success, -1 indicating a failure
47:  */
48: int Spa1Msg::setMsgSize(size_t size)
49: {
50:     int status = -1;
51:     if (size < MAX_SPA1_MSG_SIZE && size >= 0)
52:     {
53:         msgSize = size;
54:         status = 0;
55:     }
56:     return status;
57: }

```

File: sdm/Spa1Manager/Spa1Msg.h

```
1: #ifndef __SPA1_MSG_H__
2: #define __SPA1_MSG_H__
3:
4: #include <stdio.h>
5: #include <unistd.h>
6: #include <string.h>
7: #include "../common/message_defs.h"
8: #define MAX_SPA1_MSG_SIZE 256
9:
10: /**
11:  * @class Spa1Msg
12:  *
13:  * @brief Defines the Spa1Msg object for use by the SPA-1 Queue objects
14:  *
15:  * @author Bryan Hansen
16:  * @date 12/03/2009
17:  */
18: class Spa1Msg
19: {
20: public:
21:     Spa1Msg(int size);
22:     ~Spa1Msg();
23:
24:     int getMsgSize();
25:     int setMsgSize(size_t size);
26:
27:     ///Pointer to the next Spa1Msg when used in a list
28:     Spa1Msg* next;
29:
30:     ///Static array containing the message
31:     unsigned char msg[257];
32:
33: private:
34:     ///The size of the current msg
35:     int msgSize;
36: };
37:
38: #endif
```

File: sdm/Spa1Manager/Spa1Queue.cpp

```
1: #include "Spa1Queue.h"
2:
3: extern int debug;
4:
5: /**
6:  * Constructor that creates and initializes a new Spa1Queue object
7:  */
8: Spa1Queue::Spa1Queue()
9: {
10:     head = NULL;
11:     tail = NULL;
12:     msgsInQueue = 0;
13: }
14:
15: /**
16:  * Function to retrieve the next message in the queue
17:  *
18:  * @param buf The output buffer the message will be written into
19:  * @param bufSize The size of the output buffer
20:  *
21:  * @return An int containing the size of the msg read into the output buffer,
22:  *         -1 indicates that an error condition has occurred
23:  */
24: int Spa1Queue::getMsg(**Output param**/unsigned char* buf, size_t bufSize)
25: {
26:     Spa1Msg* curMsg = head;
27:     int result;
28:     if (curMsg != NULL) //Check to make sure there is a msg in the queue
29:     {
30:         if (bufSize < (unsigned int)curMsg->getMsgSize()) //Check to make sure output buffer is large
31:             enough
32:             {
33:                 printf("Spa1Queue: Error, output buffer not large enough for msg \n");
34:                 result = -1;
35:             }
36:         else
37:         {
38:             memcpy(buf, curMsg->msg, curMsg->getMsgSize()); //Copy msg into output buffer
39:             if (head == tail) //If only one msg in queue
```



```

39:     {
40:         tail = NULL;
41:     }
42:     head = head->next; //Move the head of the list forward
43:     msgInQueue--;
44:     result = curMsg->getMsgSize(); //Set size of data for return
45:
46:     delete curMsg; // delete the space
47: }
48: }
49: else
50: {
51:     result = 0; //No bytes to get
52: }
53:
54:
55: return result;
56: }
57:
58: /**
59:  * Function to place a message into the queue
60:  *
61:  * @param buf The buffer containing the message to be added to the queue
62:  * @param bufSize The size of the message to be added in bytes
63:  *
64:  * @return An int to notify success or failure - 0 indicating success, -1 for failure
65:  */
66: int Spa1Queue::putMsg(unsigned char* buf, size_t size)
67: {
68:     int result = -1;
69:
70:     if (size <= BUFSIZE) //Check if msg is larger than the max SDM Msg size
71:     {
72:         Spa1Msg* newMsg = new Spa1Msg(size);
73:         memcpy(newMsg->msg, buf, size);
74:         newMsg->next = NULL;
75:         newMsg->setMsgSize(size);
76:
77:         if (tail != NULL)
78:         {
79:             tail->next = newMsg;

```

```

80:     tail = newMsg;
81: }
82: else //Empty queue
83: {
84:     head = newMsg;
85:     tail = newMsg;
86: }
87: msgsInQueue++;
88: result = 0;
89: }
90: return result;
91: }
92:
93: /**
94: * Function to return the number of messages currently in the queue
95: *
96: * @return An int containing the number of msgs currently in the queue
97: */
98: int Spa1Queue::getNumMsgs()
99: {
100:     return msgsInQueue;
101: }

```

File: sdm/Spa1Manager/Spa1Queue.h

```
1: #ifndef __SPA1_QUEUE_H__
2: #define __SPA1_QUEUE_H__
3:
4: #include <unistd.h>
5: #include <string.h>
6: #include "Spa1Msg.h"
7: #include "../common/message_defs.h"
8:
9: /**
10: * @class Spa1Queue
11: *
12: * @brief Defines the Spa1Queue object for use by the SPA-1 Sensor Manager
13: *
14: * The Spa1Queue is used by the SPA-1 Sensor manager to queue up incoming and
15: * outgoing messages for each ASIM.
16: *
17: * @author Bryan Hansen
18: * @date 12/03/2009
19: */
20: class Spa1Queue
21: {
22: public:
23:     Spa1Queue();
24:     int getMsg(unsigned char* buf, size_t bufSize);
25:     int putMsg(unsigned char* buf, size_t size);
26:     int getNumMsgs();
27:
28: private:
29:     ///Pointer to the start of the queue
30:     Spa1Msg* head;
31:
32:     ///Pointer to the end of the queue
33:     Spa1Msg* tail;
34:
35:     ///Number of messages currently in the queue
36:     int msgsInQueue;
37: };
38:
39: #endif
```

File: sdm/Spa1Manager/Spa1Translator.cpp

```
1: #include "Spa1Translator.h"
2:
3: extern int debug;
4:
5: /**
6:  * Default constructor for the Spa1Translator
7:  */
8: Spa1Translator::Spa1Translator()
9: {
10: }
11:
12: /**
13:  * Function used to translate an SDM message into its SPA-1 equivalent.
14:  *
15:  * This function is mainly responsible to act as the public gateway to dish off
16:  * the work to the appropriate private member function.
17:  *
18:  * @param inBuf A buffer containing a marshalled SDM message
19:  * @param outBuf The buffer which will have the SPA-1 message written into it
20:  * @param outBufSize The size of the output buffer
21:  *
22:  * @return An int containing the size in bytes of the SPA-1 message written into outBuf
23:  */
24: int Spa1Translator::translateToSpa1(unsigned char* inBuf, unsigned char* outBuf, size_t outBufSize)
25: {
26:     int msgSize = 0;
27:     switch(inBuf[0])
28:     {
29:         case SDM_Serreqst:
30:             msgSize = translateSerreqst(inBuf, outBuf, outBufSize);
31:             break;
32:         case SDM_Command:
33:             msgSize = translateCommand(inBuf, outBuf, outBufSize);
34:             break;
35:         case SDM_Subreqst:
36:             msgSize = translateSubreqst(inBuf, outBuf, outBufSize);
37:             break;
38:         case SDM_Tat:
39:             msgSize = translateTat(inBuf, outBuf, outBufSize);
```

```

40:     break;
41: }
42: return msgSize;
43: }
44:
45: /**
46: * Function used to translate an SDMSerreqst into a SPA-1 Command Msg
47: *
48: * @param inBuf A buffer containing a marshalled SDM message
49: * @param outBuf The buffer which will have the SPA-1 message written into it
50: * @param outBufSize The size of the output buffer
51: *
52: * @return An int containing the size in bytes of the SPA-1 message written into outBuf
53: */
54: int Spa1Translator::translateSerreqst(unsigned char* inBuf, unsigned char* outBuf, size_t
outBufSize)
55: {
56: SDMSerreqst serReqstMsg;
57: size_t totalSize;
58: unsigned short msgSize;
59:
60: serReqstMsg.Unmarshal((const char*)inBuf);
61:
62: totalSize = SPA1_HEADER_SIZE + SECONDARY_HEADER_LEN + serReqstMsg.length;//3 Byte
header + 2 bytes for int/msg id + data length
63: msgSize = totalSize - SPA1_HEADER_SIZE;          //Length in msg sent to ASIM does not include
header
64:
65: if(totalSize > outBufSize)
66: {
67:     printf("Spa1Manager: Error, output buffer not large enough to hold translated Spa1 serreqst msg
\n");
68:     return -1;
69: }
70:
71: outBuf[0] = SPA1_ASIM_COMMAND;
72: SDM_htons(msgSize); //Make sure the length is little endian
73: memcpy(&outBuf[1], &msgSize, sizeof(msgSize));
74: outBuf[3] = serReqstMsg.command_id.getInterface();
75: outBuf[4] = serReqstMsg.command_id.getMessage();
76: memcpy(&outBuf[5], serReqstMsg.data, serReqstMsg.length);

```

```

77: debug_f(2, "Spa1Manager: Translated SDMSerreqst msg destined for ASIM %hu \n",
serReqstMsg.destination.getSensorID() & 0x00ff);
78:
79: return totalSize;
80: }
81:
82: /**
83: * Function used to translate an SDMCommand into a SPA-1 Command Msg
84: *
85: * @param inBuf A buffer containing a marshalled SDM message
86: * @param outBuf The buffer which will have the SPA-1 message written into it
87: * @param outBufSize The size of the output buffer
88: *
89: * @return An int containing the size in bytes of the SPA-1 message written into outBuf
90: */
91: int Spa1Translator::translateCommand(unsigned char* inBuf, unsigned char* outBuf, size_t
outBufSize)
92: {
93: SDMCommand commandMsg;
94: size_t totalSize;
95: unsigned short msgSize;
96:
97: commandMsg.Unmarshal((const char*)inBuf);
98:
99: totalSize = SPA1_HEADER_SIZE + SECONDARY_HEADER_LEN + commandMsg.length; //3
Byte header + 2 bytes for int/msg id + data length
100: msgSize = totalSize - SPA1_HEADER_SIZE; //Length in msg sent to ASIM does not
include header
101:
102: if(totalSize > outBufSize)
103: {
104: printf("Spa1Manager: Error, output buffer not large enough to hold translated Spa1 command
msg \n");
105: return -1;
106: }
107:
108: outBuf[0] = SPA1_ASIM_COMMAND;
109: SDM_htons(msgSize); //Make sure the length is little endian
110: memcpy(&outBuf[1], &msgSize, sizeof(msgSize));
111: outBuf[3] = commandMsg.command_id.getInterface();
112: outBuf[4] = commandMsg.command_id.getMessage();
113: memcpy(&outBuf[5], commandMsg.data, commandMsg.length);

```

```

114:  debug_f(2, "Spa1Manager: Translated SDMCommand msg destined for ASIM %hu \n",
commandMsg.destination.getSensorID() & 0x00ff);
115:
116:  return totalSize;
117: }
118:
119: /**
120: * Function used to translate an SDMSubreqst into a SPA-1 Data Request Msg
121: *
122: * @param inBuf A buffer containing a marshalled SDM message
123: * @param outBuf The buffer which will have the SPA-1 message written into it
124: * @param outBufSize The size of the output buffer
125: *
126: * @return An int containing the size in bytes of the SPA-1 message written into outBuf
127: */
128: int Spa1Translator::translateSubreqst(unsigned char* inBuf, unsigned char* outBuf, size_t
outBufSize)
129: {
130:     SDMSubreqst subReqstMsg;
131:     size_t totalSize;
132:     unsigned short msgSize;
133:
134:     subReqstMsg.Unmarshal((const char*)inBuf);
135:
136:     totalSize = SPA1_HEADER_SIZE + SECONDARY_HEADER_LEN; //3 Byte header + 2
bytes for int/msg id
137:     msgSize = totalSize - SPA1_HEADER_SIZE; //Length in msg sent to ASIM does not
include header
138:
139:     if(totalSize > outBufSize)
140:     {
141:         printf("Spa1Manager: Error, output buffer not large enough to hold translated Spa1 request
data msg \n");
142:         return -1;
143:     }
144:
145:     outBuf[0] = SPA1_ASIM_DATAREQ;
146:     SDM_htons(msgSize); //Make sure the length is little endian
147:     memcpy(&outBuf[1], &msgSize, sizeof(msgSize));
148:     outBuf[3] = subReqstMsg.msg_id.getInterface();
149:     outBuf[4] = subReqstMsg.msg_id.getMessage();
150:

```

```

151:     debug_f(2, "Spa1Manager: Translated SDMSubreqst msg destined for ASIM %hu \n",
subReqstMsg.source.getSensorID() & 0x00ff);
152:
153:     return totalSize;
154: }
155:
156: /**
157:  * Function used to package up a SPA-1 Cancel Data Message
158:  *
159:  * @param intId The interface id of the data to cancel
160:  * @param msgId The message id of the data to cancel
161:  * @param outBuf The buffer which will have the SPA-1 message written into it
162:  * @param outBufSize The size of the output buffer
163:  *
164:  * @return An int containing the size in bytes of the SPA-1 message written into outBuf
165:  */
166: int Spa1Translator::packageCancelData(unsigned char intId, unsigned char msgId, unsigned char*
outBuf, size_t outBufSize)
167: {
168:     size_t totalSize;
169:     unsigned short msgSize;
170:     totalSize = SPA1_HEADER_SIZE + SECONDARY_HEADER_LEN; //3 Byte header + 2
bytes for int/msg id
171:     msgSize = totalSize - SPA1_HEADER_SIZE;
172:
173:     if(totalSize > outBufSize)
174:     {
175:         printf("Spa1Manager: Error, output buffer not large enough to hold translated Spa1 cancel
data msg \n");
176:         return -1;
177:     }
178:
179:     outBuf[0] = SPA1_ASIM_CANCEL_DATA;
180:     SDM_htons(msgSize); //Make sure the length is little endian
181:     memcpy(&outBuf[1], &msgSize, sizeof(msgSize));
182:     outBuf[3] = intId;
183:     outBuf[4] = msgId;
184:
185:     return totalSize;
186: }
187:
188: /**

```



```
189: * Function used to translate an SDMTat msg into a SPA-1 Tat Msg
190: *
191: * @param inBuf A buffer containing a marshalled SDM message
192: * @param outBuf The buffer which will have the SPA-1 message written into it
193: * @param outBufSize The size of the output buffer
194: *
195: * @return An int containing the size in bytes of the SPA-1 message written into outBuf
196: */
197: int Spa1Translator::translateTat(unsigned char* inBuf, unsigned char* outBuf, size_t outBufSize)
198: {
199:     //TODO
200:     return 0;
201: }
```

File: sdm/Spa1Manager/Spa1AsimTable.cpp

```
1: #include "Spa1AsimTable.h"
2:
3: extern int debug;
4:
5: /**
6:  * Default constructor for the Spa1AsimTable object
7:  *
8:  */
9: Spa1AsimTable::Spa1AsimTable() : head(NULL), tail(NULL), nextSensorIndex(2), asimCount(0)
10: {
11:     memset (addressSpace, 0, 128);
12:     memset (addressSpace, 1, 16); //First 16 addresses are reserved by the I2C standard or by us for
future use
13: pthread_mutex_init(&asimTableMutex, NULL);
14: }
15:
16: /**
17:  * Checks to see if an I2C address is currently in use
18:  *
19:  * @param address The I2C address to check for availability
20:  *
21:  * @return A bool True meaning the address is used, false means it is available
22:  */
23: bool Spa1AsimTable::checkAddressUsed(unsigned char address)
24: {
25:     return addressSpace[address];
26: }
27:
28: /**
29:  * Adds a Spa1Asim to the table
30:  *
31:  * @param asim A pointer to the new ASIM to add to the table
32:  */
33: void Spa1AsimTable::addAsim(Spa1Asim* asim)
34: {
35:     pthread_mutex_lock(&asimTableMutex);
36:     addressSpace[asim->getI2cAddress()] = 1;
37:     asim->setSensorIndex(nextSensorIndex);
38:
```

```

39: nextSensorIndex++;
40:
41: if (head == NULL)
42: {
43:     head = asim;
44:     tail = head;
45: }
46: else
47: {
48:     tail->next = asim;
49:     tail = tail->next;
50: }
51: asimCount++;
52: pthread_mutex_unlock(&asimTableMutex);
53: }
54:
55:
56: /**
57:  * Removes a Spa1Asim from the table
58:  *
59:  * @param address The I2C address of the ASIM to remove
60:  */
61: void Spa1AsimTable::removeAsim(unsigned char address)
62: {
63:     Spa1Asim* toRemove;
64:     Spa1Asim* cur;
65:
66:     addressSpace[address] = 0;
67:
68: //No mutexes here as this is only called when the mutex is already held for the table
69: if (head != NULL)
70: {
71:     if (head->getI2cAddress() == address)
72:     {
73:         toRemove = head;
74:         head = head->next;
75:         delete toRemove;
76:         asimCount--;
77:         return;
78:     }
79:

```

```

80:   for (cur = head; cur->next != tail; cur = cur->next)
81:   {
82:       if (cur->next->getI2cAddress() == address)
83:       {
84:           toRemove = cur->next;
85:           cur->next = cur->next->next;
86:           delete toRemove;
87:           asimCount--;
88:           return;
89:       }
90:   }
91:
92:   if (tail->getI2cAddress() == address)
93:   {
94:       toRemove = tail;
95:       tail = cur;
96:       cur->next = NULL;
97:       delete toRemove;
98:       asimCount--;
99:   }
100: }
101: }
102:
103: /**
104:  * Gets a Spa1Asim from the table by its Sensor Id
105:  *
106:  * @param sensorId The 4 byte SDM sensorId as assigned by the DataManager
107:  *
108:  * @return A Spa1Asim* which points to the requested Spa1Asim object
109:  */
110: Spa1Asim* Spa1AsimTable::getAsimBySensorId(unsigned long sensorId)
111: {
112:     unsigned short sensorIndex = sensorId & 0x00ff; //Chop off top two bytes used by the SDM
113:     return getAsimBySensorIndex(sensorIndex);
114: }
115:
116: /**
117:  * Gets a Spa1Asim from the table by its Sensor Index
118:  *
119:  * @param sensorId The 2 byte sensor index as assigned by the Spa1Manager
120:  *

```

```

121: * @return A Spa1Asim* which points to the requested Spa1Asim object
122: */
123: Spa1Asim* Spa1AsimTable::getAsimBySensorIndex(unsigned short sensorIndex)
124: {
125:     Spa1Asim* cur;
126:
127:     pthread_mutex_lock(&asimTableMutex);
128:     for (cur = head; cur != NULL; cur = cur->next)
129:     {
130:         if(cur->getSensorIndex() == sensorIndex)
131:         {
132:             pthread_mutex_unlock(&asimTableMutex);
133:             return cur;
134:         }
135:     }
136:     pthread_mutex_unlock(&asimTableMutex);
137:
138:     return NULL;
139: }
140:
141: /**
142: * Gets a Spa1Asim from the table by the GUID it used to gain I2C arbitration
143: *
144: * @param guid The 4 byte buid used to gain arbitration on the I2C bus
145: *
146: * @return A Spa1Asim* which points to the requested Spa1Asim object
147: */
148: Spa1Asim* Spa1AsimTable::getAsimByGuid(int guid)
149: {
150:     Spa1Asim* cur;
151:
152:     pthread_mutex_lock(&asimTableMutex);
153:     for (cur = head; cur != NULL; cur = cur->next)
154:     {
155:         if(cur->getGuid() == guid)
156:         {
157:             pthread_mutex_unlock(&asimTableMutex);
158:             return cur;
159:         }
160:     }
161:     pthread_mutex_unlock(&asimTableMutex);

```

```

162:
163:     return NULL;
164: }
165:
166: /**
167:  * Function to write data out to the ASIMs during the round robin phase
168:  *
169:  * Sending data to ASIMs in this round robin fashion should prevent any ASIMs
170:  * from being ignored or prioritized
171:  */
172: void Spa1AsimTable::sendToAsims()
173: {
174:     unsigned char buf[MAX_ASIM_MSG_SIZE];
175:     int msgSize;
176:
177:     pthread_mutex_lock(&asimTableMutex);
178:     for(Spa1Asim* cur = head; cur != NULL; cur = cur->next)
179:     {
180:         if(cur->sendToQueue.getNumMsgs() > 0)
181:         {
182:             msgSize = cur->sendToQueue.getMsg(buf, MAX_ASIM_MSG_SIZE); //Retreive msg
from queue
183:             if(msgSize > 0)
184:             {
185:                 cur->sendToAsim(buf, msgSize);
186:                 debug_f(2, "Spa1Manager: Sent %i byte msg to ASIM %i \n", msgSize, cur-
>getSensorIndex());
187:             }
188:             else
189:             {
190:                 printf("Spa1Queue: Error retrieving msg from queue to send to ASIM \n");
191:             }
192:         }
193:     }
194:     pthread_mutex_unlock(&asimTableMutex);
195: }
196:
197: /**
198:  * Function to read data from the ASIMs during the round robin phase
199:  *
200:  * Reading data from ASIMs in this round robin fashion should prevent any ASIMs

```

```

201: * from being ignored or prioritized
202: */
203: void Spa1AsimTable::recvFromAsims()
204: {
205:     unsigned char buf[MAX_ASIM_MSG_SIZE];
206:     int msgSize;
207:     SDMCancelxTEDS cancelxTEDSMsg;
208:
209:     pthread_mutex_lock(&asimTableMutex);
210:     for(Spa1Asim* cur = head; cur != NULL; cur = cur->next)
211:     {
212:         msgSize = cur->recvFromAsim(buf, MAX_ASIM_MSG_SIZE);
213:
214:         if(msgSize > 0) //Msg Received
215:         {
216:             cur->recvFromQueue.putMsg(buf, msgSize); //Add msg to ASIM's queue
217:             debug_f(2, "Spa1Manager: Received Msg from ASIM %i and added it to it's outgoing
queue \n", cur->getSensorIndex());
218:         }
219:         else if(msgSize == -1) //ASIM did not respond
220:         {
221:             cur->incrementFailCount();
222:             debug_f(2, "Spa1Manager: ASIM %i unresponsive, fail count: %i \n", cur-
>getSensorIndex(), cur->getFailCount());
223:             if(cur->getFailCount() > MAX_ASIM_FAILURES)
224:             {
225:                 cancelxTEDSMsg.source.setPort(PORT_SPA1_MANAGER);
226:                 cancelxTEDSMsg.source.setSensorID(cur->getSensorIndex());
227:                 cancelxTEDSMsg.Send();
228:                 debug_f(2, "Spa1Manager: Canceling xTEDS and removing ASIM %i from ASIM
table \n", cur->getSensorIndex());
229:                 removeAsim(cur->getI2cAddress());
230:             }
231:         }
232:     }
233:     pthread_mutex_unlock(&asimTableMutex);
234: }
235:
236: /**
237: * Function called periodically to check the ASIMs to see if they have any
238: * data or status messages queued up in their output buffer that need to be
239: * handled.

```

```

240: *
241: * If a data message is pulled from an ASIMs queue, the message is published. If
242: * there are no subscribers for the given message then a cancel data message is sent
243: * to the ASIM so that it will stop producing that data.
244: * Status messages are examined and if an error bit is set a message is currently printed
245: * to the screen.
246: */
247: void Spa1AsimTable::checkMsgsFromAsim()
248: {
249:     unsigned char buf[MAX_ASIM_MSG_SIZE];
250:     Spa1Translator translator;
251:     short msgSize;
252:
253:     pthread_mutex_lock(&asimTableMutex);
254:     for(Spa1Asim* cur = head; cur != NULL; cur = cur->next)
255:     {
256:         if(cur->recvFromQueue.getNumMsgs() > 0) //Msgs to be looked at or sent out
257:         {
258:             debug_f(2, "Spa1Manager: %i msgs in ASIM %i's outgoing queue \n", cur-
>recvFromQueue.getNumMsgs(), cur->getSensorIndex());
259:             cur->recvFromQueue.getMsg(buf, MAX_ASIM_MSG_SIZE);
260:             memcpy(&msgSize, &buf[1], 2);
261:             msgSize = SDM_ntohs(msgSize);
262:             switch(buf[0])
263:             {
264:                 case 'D':
265:                     if(cur->subscriptions.Publish(buf[3], buf[4], (char*)&buf[5], msgSize))
266:                     {
267:                         debug_f(2, "Spa1Manager: Published data from ASIM %i for msg (%i,%i)
\n", cur->getSensorIndex(), buf[3], buf[4]);
268:                     }
269:                     else
270:                     {
271:                         debug_f(2, "Spa1Manager: Received data from ASIM %i for msg (%i,%i),
but no subscribers \n", cur->getSensorIndex(), buf[3], buf[4]);
272:                         msgSize = translator.packageCancelData(buf[3], buf[4], buf,
MAX_ASIM_MSG_SIZE);
273:                         cur->sendToAsim(buf, msgSize); //Send cancel data msg to ASIM
274:                     }
275:                     break;
276:                 case 'S':

```



```

277:             debug_f(2, "Spa1Manager: Received Status from ASIM %i: 0x%02x \n", cur-
>getSensorIndex(), buf[3]);
278:
279:             if(buf[3] & 0x80) //Error bit(s) set //TODO: Are there any actions which should
be taken if one of these errors is reported?
280:             {
281:                 if(buf[3] & 0x40)
282:                 {
283:                     debug_f(1, "Spa1Manager: ASIM %i reported an illegal opcode error
\n", cur->getSensorIndex());
284:                 }
285:                 if(buf[3] & 0x20)
286:                 {
287:                     debug_f(1, "Spa1Manager: ASIM %i reported an unknown interface id
and/or msg id \n", cur->getSensorIndex());
288:                 }
289:                 if(buf[3] & 0x10)
290:                 {
291:                     debug_f(1, "Spa1Manager: ASIM %i reported a self-test failure \n", cur-
>getSensorIndex());
292:                 }
293:             }
294:             else if(buf[3] != 0) //Operational status bits set
295:             {
296:                 //TBD
297:             }
298:             break;
299:             default:
300:                 break;
301:         }
302:     }
303: }
304: pthread_mutex_unlock(&asimTableMutex);
305: }
306:
307: /**
308:  * Cancels the SDM registrations for all registered SPA-1 ASIMs
309:  *
310:  */
311: void Spa1AsimTable::cancelSdmRegistrations()
312: {
313:     SDMCancelxTEDS cancelxTEDSMsg;

```

```

314:   cancelxTEDSMsg.source.setPort(PORT_SPA1_MANAGER);
315:
316:   for(Spa1Asim* cur = head; cur != NULL; cur = cur->next)
317:   {
318:       cancelxTEDSMsg.source.setSensorID(cur->getSensorIndex());
319:       cancelxTEDSMsg.Send();
320:       debug_f(2, "Spa1Manager: Canceling xTEDS for ASIM %i \n", cur->getSensorIndex());
321:   }
322: }
323:
324: /**
325:  * Utility function to print out the asim table along with details about each asim
326:  *
327:  */
328: void Spa1AsimTable::printAsimData()
329: {
330:     printf(" \n*****ASIM Table - size: %i***** \n", asimCount);
331:     for(Spa1Asim* cur = head; cur != NULL; cur = cur->next)
332:     {
333:         printf(" \tASIM %i - RegState: %i, I2C Addr: 0x%x, GUID: %i, FailCount: %i \n", cur-
>getSensorIndex(), cur->getRegState(), cur->getI2cAddress(), cur->getGuid(), cur->getFailCount());
334:     }
335:     printf("***** \n \n");
336: }
337:
338: /**
339:  * Returns the number of currently registered SPA-1 ASIMs
340:  *
341:  * @return An int containing the number of registered SPA-1 ASIMs
342:  */
343: int Spa1AsimTable::getAsimCount()
344: {
345:     return asimCount;
346: }

```

File: sdm/Spa1Manager/Makefile

```
1: include ../Makefile.common
2: include ../$(MAKEFILE_DEFS)
3:
4: .PHONY:    all clean distclean
5: #CXX=/opt/gumstix/build_arm_nofpu/staging_dir/bin/arm-linux-uclibcgnueabi-g++
6:
7:
8: all: Spa1Manager
9:
10: Spa1Manager:  Spa1Manager.o  Spa1Asim.o  Spa1Queue.o  Spa1Msg.o  Spa1AsimTable.o
Spa1Translator.o
11: $(CXX) $(CXXFLAGS) -L../common -static -o $@ $^ $(BOOSTFLAGS) -lSDM -lpthread
12:
13: Spa1Manager.o: Spa1Manager.cpp
14: $(CXX) $(CXXFLAGS) -c $<
15:
16: Spa1Asim.o:    Spa1Asim.cpp Spa1Asim.h Spa1Queue.o
17: $(CXX) $(CXXFLAGS) -c -o $@ $<
18:
19: Spa1Queue.o: Spa1Queue.cpp Spa1Queue.h Spa1Msg.o
20: $(CXX) $(CXXFLAGS) -c -o $@ $<
21:
22: Spa1Msg.o: Spa1Msg.cpp
23: $(CXX) $(CXXFLAGS) -c -o $@ $<
24:
25: Spa1AsimTable.o: Spa1AsimTable.cpp
26: $(CXX) $(CXXFLAGS) -c -o $@ $<
27:
28: Spa1Translator.o: Spa1Translator.cpp
29: $(CXX) $(CXXFLAGS) -c -o $@ $<
30:
31: clean:
32: rm -f *.o *~ *.out
33:
34: distclean: clean
35: rm -f Spa1Manager
```

File: sdm/Spa1Manager/Spa1Manager.cpp

```
1: #include "Spa1Manager.h"
2:
3: /**
4:  * Main routine for the SPA-1 Manager
5:  *
6:  * Parses command line options and starts the signal handling, sdmLisener, and
7:  * i2cComm threads.
8:  *
9:  */
10: int main(int argc, char** argv)
11: {
12:     MessageManager msgManager;
13:     pthread_attr_t threadAttr;
14:     int option = 0;
15:
16:     DataManager.setAddress(inet_addr("127.0.0.1"));
17:     DataManager.setPort(PORT_DM);
18:
19:     while(option != -1)
20:     {
21:         option = getopt(argc,argv,"g:d:v");
22:         switch (option)
23:         {
24:             case 'g':
25:                 debug = atoi(optarg);
26:                 break;
27:             case 'd':
28:                 DataManager.setAddress(inet_addr(optarg));
29:                 if (((long)DataManager.getAddress()) == -1)
30:                 {
31:                     printf("Error in DM address. Format is -d[IP Addr in string notation] \n");
32:                     return 0;
33:                 }
34:                 DataManager.setPort(PORT_DM);
35:                 break;
36:             case 'v':
37:                 printf("SDM Version: %s Repo Rev: %i \n", SDM_VERSION, REVISION_NUMBER);
38:                 return 0;
39:             default:
```

```

40:         break;
41:     }
42: }
43:
44: #ifndef WIN32
45: sigset_t sigSet;
46: sigemptyset(&sigSet);
47: sigaddset(&sigSet, SIGINT);
48: pthread_sigmask(SIG_BLOCK, &sigSet, NULL);
49:
50: if(pthread_create(&sigHandler, NULL, &signalHandler, NULL) != 0)
51: {
52:     printf("SPA-1 Manager: Could not start signal handler thread. \n");
53: }
54: #else
55: signal(SIGINT, signalHandler);
56: #endif
57:
58: msgManager.Async_Init(PORT_SPA1_MANAGER);
59:
60: if(sdmRegister(&msgManager) != 0)
61: {
62:     printf("SPA-1 Manager: Error Registering with SDM. \n");
63:     return -1;
64: }
65:
66: pthread_attr_init(&threadAttr);
67: pthread_attr_setstacksize(&threadAttr, THREAD_STACK_SIZE);
68:
69: if (pthread_create(&sdmListenThread, &threadAttr, &sdmListener, &msgManager) < 0)
70: {
71:     printf("SPA-1 Manager: Error creating SDM Listener thread \n");
72: }
73:
74: if (pthread_create(&i2cThread, &threadAttr, &i2cComm, NULL) < 0)
75: {
76:     printf("SPA-1 Manager: Error creating I2C Comm thread \n");
77: }
78:
79: asimMonitor();
80:

```

```

81: pthread_join(sdmListenThread, NULL);
82: pthread_join(i2cThread, NULL);
83:
84: return 0;
85: }
86:
87: /**
88:  * Function used to register the SPA-1 Manager with the SDM
89:  *
90:  * @param msgManager A pointer to the Message Manager being used by the SPA-1 Manager
91:  *
92:  * @return An int indicating the outcome of the registration, 0 for success, -1 otherwise
93:  */
94: int sdmRegister(MessageManager* msgManager)
95: {
96:     SDMHHello helloMsg;
97:     SDMAck ackMsg;
98:     SDMID idMsg;
99:     bool isRegistered = false;
100:    bool ackReceived = false;
101:    char buf[BUFSIZE];
102:    double timeout = 5.0;
103:    double endTime;
104:
105:    helloMsg.type = 'C';
106:    helloMsg.source.setPort(PORT_SPA1_MANAGER);
107:    helloMsg.Send();
108:    debug_f(1, "Spa1Manager: Sending SDMHHello Msg \n");
109:
110:    endTime = getCurTime() + timeout;
111:
112:    while (!isRegistered)
113:    {
114:        if (msgManager->IsReady())
115:        {
116:            #ifdef WIN32
117:                switch (msgManager->GetMsg(buf))
118:            #else
119:                switch (msgManager->GetMessage(buf))
120:            #endif
121:            {

```

```

122:     case SDM_ACK:
123:         ackReceived = true;
124:         debug_f(1, "Spa1Manager: SDMAck received \n");
125:         break;
126:     case SDM_Register:
127:         debug_f(1, "Spa1Manager: SDMRegister Received \n");
128:         registerxTEDS();
129:         endTime = getCurTime() + timeout;
130:         break;
131:     case SDM_ID:
132:         isRegistered = true;
133:         debug_f(1, "Spa1Manager: SDMID received \n");
134:         break;
135:     default:
136:         break;
137: }
138: }
139: else
140: {
141:     if (!ackReceived && getCurTime() >= endTime)
142:     {
143:         helloMsg.Send();
144:         endTime = getCurTime() + timeout;
145:         debug_f(1, "Spa1Manager: Resending SDMHHello Msg \n");
146:     }
147:     else if (ackReceived && !isRegistered && getCurTime() >= endTime)
148:     {
149:         registerxTEDS();
150:         endTime = getCurTime() + timeout;
151:     }
152: }
153: if(termination)
154: {
155:     return -1;
156: }
157: usleep(10000);
158: }
159: debug_f(1, "Spa1Manager: Successully Registered with Data Manager \n");
160: return 0;
161: }
162:

```

```

163: /**
164: * Function used to register the SPA-1 Manager's xTEDS
165: *
166: */
167: void registerxTEDS()
168: {
169:     SDMxTEDS xTEDSMsg;
170:     xTEDSMsg.source.setSensorID(1);
171:     xTEDSMsg.source.setPort(PORT_SPA1_MANAGER);
172:     strcpy(xTEDSMsg.xTEDS, spa1ManagerxTEDS);
173:     xTEDSMsg.Send();
174:     debug_f(1, "Spa1Manager: Sending xTEDS to Data Manager \n");
175: }
176:
177: /**
178: * Thread that listens for incoming SDM messages and responds appropriately
179: *
180: * @param args A pointer the the SPA-1 Manager's Message Manager
181: *
182: */
183: void* sdmListener(void* args)
184: {
185:     MessageManager* msgManager = (MessageManager*)args;
186:     char buf[BUFSIZE];
187:
188:     while (!termination)
189:     {
190:         if (msgManager->IsReady())
191:         {
192: #ifdef WIN32
193:             switch (msgManager->GetMsg(buf))
194: #else
195:             switch (msgManager->GetMessage(buf))
196: #endif
197:             {
198:                 case SDM_Serreqst:
199:                     handleSerreqstMsg(buf);
200:                     break;
201:                 case SDM_Command:
202:                     handleCommandMsg(buf);
203:                     break;

```



```

204:     case SDM_Subreqst:
205:         handleSubreqstMsg(buf);
206:         break;
207:     case SDM_Deletesub:
208:         handleDeletesubMsg(buf);
209:         break;
210:     case SDM_Tat:
211:         // TODO:
212:         break;
213:     case SDM_Ready:
214:         break;
215:     case SDM_RegInfo:
216:         break;
217:     case SDM_Data:
218:         break;
219:     case SDM_ACK:
220:         handleAckMsg(buf);
221:         break;
222:     case SDM_Register:
223:         handleRegisterMsg(buf);
224:         break;
225:     case SDM_ID:
226:         handleIdMsg(buf);
227:         break;
228:     default:
229:         break;
230: }
231: }
232: usleep(10000);
233: }
234: return NULL;
235: }
236:
237: /**
238:  * Handles all the I2C communication between the SPA-1 Manager and the
239:  * SPA-1 ASIMs
240:  *
241:  * Opens the I2C file descriptor and starts the round robin communication.
242:  * The round robin consists of three phases, writing to ASIMs, receiving from
243:  * ASIMs, and searching for new ASIMs. The first two phase occur every iteration
244:  * while searching for new ASIMs occurs only once every 256 iterations, essentially

```

```

245: * seaching for new ASIMs on the bus 4x per second.
246: */
247: void* i2cComm(void* args)
248: {
249:     unsigned char highestKnownAddress = DEFAULT_ADDRESS; // no known addresses
250:     unsigned int endTime = 0;
251:     debug_f(2, "Spa1Manager: i2c Comm Thread started \n");
252:     busFd = open("/dev/i2c-0", O_RDWR);
253:
254:     if (busFd == -1)
255:     {
256:         printf("Spa1Manager: Unable to open /dev/i2c-0, exiting... \n");
257:         termination = 1;
258:         cancelSdmRegistrations();
259:         return NULL;
260:     }
261:
262:     // Start round robin
263:     debug_f(1, "Spa1Manager: Starting round robin \n");
264:
265:     while (!termination)
266:     {
267:         asimTable.sendToAsims(); //Round robin write
268:
269:         usleep(1000); //Keep loop from running too much and give ASIMs a little time to respond
270:
271:         asimTable.recvFromAsims(); //Round robin read
272:
273:         if(getCurTime() > endTime)
274:         {
275:             endTime = getCurTime() + 1;
276:             highestKnownAddress = searchForNewAsims(highestKnownAddress);
277:             debug_f(3, "highestKnownAddress == %x \n", highestKnownAddress);
278:             if(debug > 3)
279:             {
280:                 asimTable.printAsimData();
281:             }
282:         }
283:     }
284:
285:     return NULL;

```

```

286: }
287:
288: /**
289: * Function which checks periodically checks each ASIM for any outgoing data
290: * or received status messages.
291: *
292: * Messages are checked at roughly 1
293: * @param args A pointer the the SPA-1 Manager's Message Manager
294: *
295: */
296: void asimMonitor()
297: {
298:     while(!termination)
299:     {
300:         asimTable.checkMsgsFromAsim(); //Checks for msgs from ASIMs, publishes data msgs and
inspects status msgs
301:         usleep(1000);
302:     }
303: }
304:
305:
306: /**
307: * Searches the address space for any unregistered ASIMs. When a new ASIM is
308: * found the registration process is initiated for that ASIM.
309: *
310: * @param highestKnownAddress The highest i2c address currently registered with the
Spa1Manager
311: *
312: * @return An unsigned char of the updated highest known used i2c address
313: */
314: unsigned char searchForNewAsims(unsigned char highestKnownAddress)
315: {
316:     debug_f(4, "Spa1Manager: Searching for new ASIMs \n");
317:     unsigned char buf[DEFAULT_BUFFER_SIZE];
318:     unsigned char highestAddressResponse = highestKnownAddress;
319:
320:     int bytesRead = 0;
321:     pthread_t sdmRegThread;
322:     pthread_attr_t threadAttr;
323:
324:     pthread_attr_init(&threadAttr);

```

```

325: pthread_attr_setdetachstate(&threadAttr, PTHREAD_CREATE_DETACHED);
326:
327: unsigned char probeMsg[3];
328: probeMsg[0] = 'Z';
329: probeMsg[1] = 0;
330: probeMsg[2] = 0;
331:
332: for (unsigned char address = DEFAULT_ADDRESS; (address <= highestKnownAddress) ||
(address < 0x80 && bytesRead != -1); ++address)
333: {
334:     if (!asimTable.checkAddressUsed(address))
335:     {
336:         setSlaveAddress(address);
337:         write(busFd, probeMsg, 3);
338:         usleep(1000); //Give ASIM a little time to process message
339:         bytesRead = i2cRead(buf, SPA1_HEADER_SIZE + 4);
340:
341:         debug_f(3, "bytesRead == %i \n", bytesRead);
342:         if(debug >= 4)
343:         {
344:             for(int i=0; i < bytesRead; i++)
345:             {
346:                 debug_f(4, "buf[%i]: 0x%02x \n", i, buf[i]);
347:             }
348:         }
349:         if (bytesRead != -1)
350:         {
351:             int guid = 0;
352:             memcpy(&guid, &buf[3], 4);
353:
354:             debug_f(1, "Spa1Manager: Registering ASIM with GUID: %i at address 0x%x \n", guid,
address);
355:
356:             Spa1Asim* asim = new Spa1Asim(busFd, address, guid);
357:             bool success = asim->doRegistration();
358:
359:             if (success)
360:             {
361:                 debug_f(1, "Spa1Manager: Successful registration! Spawning thread to register with the
DM \n");
362:                 asimTable.addAsim(asim);

```

```

363:             debug_f(1, "Spa1Manager: Added ASIM to table, current count: %i \n",
asimTable.getAsimCount());
364:
365:             if (address > highestKnownAddress)
366:             {
367:                 highestAddressResponse = address;
368:             }
369:
370:             if(pthread_create(&sdmRegThread, &threadAttr, &regAsimWithSdm, asim) < 0) //Thread
is created detached
371:             {
372:                 printf("Spa1Manager: Error spawning thread to register ASIM with DM \n");
373:             }
374:         }
375:     else
376:     {
377:         debug_f(1, "Spa1Manager: Failed registration for ASIM with GUID: %i \n", guid);
378:     }
379: }
380: else if (address < highestKnownAddress)
381: {
382:     debug_f(2, "Spa1Manager: Detected hole in address space at address 0x%x \n", address);
383: }
384: }
385: }
386:
387: return highestAddressResponse;
388: }
389:
390:
391: /**
392: * Attempts to read data off of the i2c bus.
393: *
394: * If 0xff is read off of the bus, it is assumed that the ASIM is not yet ready to provide data
395: *
396: * @param buf The output buffer to read into
397: * @param size The size of the output buf
398: *
399: * @return An int containing the number of bytes read. -1 signifying a timeout or error condition.
400: */
401: int i2cRead(unsigned char* buf, size_t size)

```

```

402: {
403:   int bytesRead = 0;
404:   bool dataReady = false;
405:   double endTime = getCurTime() + I2C_READ_TIMEOUT;
406:
407:   while (!dataReady)
408:   {
409:     bytesRead = read(busFd, buf, size);
410:
411:     //0xff is placed onto the ASIMs output register to signify that it has no data ready
412:     if (bytesRead != -1 && buf[0] == 0xff && buf[1] == 0xff && buf[2] == 0xff)
413:     {
414:       dataReady = false;
415:     }
416:     else
417:     {
418:       dataReady = true;
419:     }
420:
421:     if (getCurTime() > endTime)
422:     {
423:       bytesRead = -1;
424:       break;
425:     }
426:   }
427:
428:   if (bytesRead == -1)
429:   {
430:     debug_f(3, "No response \n");
431:   }
432:
433:   return bytesRead;
434: }
435:
436: /**
437:  * Sets the address of the I2C slave currently being read/written to
438:  *
439:  * @param address The address to interact with on the I2C bus
440:  */
441: void setSlaveAddress(unsigned char address)
442: {

```

```

443:  debug_f(4, "Setting slave address to: %x \n", address);
444:
445: #ifndef WIN32
446:  if (ioctl(busFd, I2C_SLAVE, address) < 0)
447:  {
448:      printf("Spa1Manager: Can't use ioctl to set i2c slave address \n");
449:  }
450: #endif
451: }
452:
453: /**
454:  * Thread used to register an individual ASIM with the SDM
455:  *
456:  * @param args A pointer to the Spa1Asim object to register
457:  *
458:  */
459: void* regAsimWithSdm(void* args)
460: {
461:     Spa1Asim* asim = (Spa1Asim*)args;
462:     SDMHHello helloMsg;
463:     double endTime = 0.0;
464:     int timeout = 5;
465:     char notificationBuf[5];
466:
467:     helloMsg.source.setPort(PORT_SPA1_MANAGER);
468:     helloMsg.source.setSensorID(asim->getSensorIndex());
469:     helloMsg.Send();
470:     debug_f(1, "Spa1Manager: Sending SDMHHello for ASIM with sensorIndex: %i \n", asim-
471: >getSensorIndex());
472:     asim->setRegState(SENT_HELLO);
473:     endTime = getCurTime() + timeout;
474:
475:     while (asim->getRegState() == SENT_HELLO)
476:     {
477:         if(getCurTime() > endTime)
478:         {
479:             helloMsg.Send();
480:             debug_f(1, "Spa1Manager: Re-sending SDMHHello for ASIM with sensorIndex: %i \n",
481: asim->getSensorIndex());
482:             // reset the timeout after each timeout

```

```

482:         endTime = getCurTime() + timeout;
483:     }
484:     usleep(100000);
485: }
486:
487: while (asim->getRegState() != RECVD_REG)
488: {
489:     usleep(100000);
490: }
491:
492: registerAsimXteds(asim);
493:
494: while (asim->getRegState() != RECVD_ID)
495: {
496:     usleep(100000);
497: }
498:
499: notificationBuf[0] = 0; //0 = Registration
500: PUT_UINT(&notificationBuf[1], asim->getSensorId());
501: subscriptions.Publish(regChangeMsg, notificationBuf, 5); //Attempt to publish notification of an
ASIM registration
502:
503: debug_f(1, "Spa1Manager: ASIM %i successfully registered with the SDM \n", asim-
>getSensorIndex());
504: return NULL;
505: }
506:
507: /**
508: * Used to handle an SDMAck message received by the SPA-1 Manager
509: *
510: * When an SDMAck message is received, this function sets the
511: * registration state of the corresponding Spa1Asim object to
512: * RECVD_ACK. This allows the ASIMs registration thread to proceed.
513: *
514: * @param buf The buffer containing the received SDMAck msg
515: *
516: */
517: void handleAckMsg(char* buf)
518: {
519:     SDMAck ackMsg;
520:     Spa1Asim* asim;

```



```

521:   if(ackMsg.Unmarshal(buf) < 0)
522:   {
523:       printf("Spa1Manager: Bad SDMAck Msg received \n");
524:   }
525:
526:   asim = asimTable.getAsimBySensorId(ackMsg.error);
527:
528:   if(asim == NULL)
529:   {
530:       printf("Spa1Manager: No matching ASIM found for SDMAck with sensorIndex: %i \n",
531:       ackMsg.error);
532:       return;
533:   }
534:   debug_f(1, "Spa1Manager: SDMAck Received for ASIM %i \n", asim->getSensorIndex());
535:   asim->setRegState(RECVD_ACK);
536: }
537:
538: /**
539:  * Used to handle an SDMRegister message received by the SPA-1 Manager
540:  *
541:  * When an SDMRegister message is received, this function sets the
542:  * corresponding ASIMs registration state to RECVD_REG. This causes
543:  * the ASIM's registration thread to continue and send the ASIM's xTEDS.
544:  *
545:  * @param buf The buffer containing the received SDMRegister msg
546:  *
547:  */
548: void handleRegisterMsg(char* buf)
549: {
550:     SDMRegister registerMsg;
551:     Spa1Asim* asim;
552:     if(registerMsg.Unmarshal(buf) < 0)
553:     {
554:         printf("Spa1Manager: Bad SDMRegister Msg received \n");
555:     }
556:
557:     asim = asimTable.getAsimBySensorIndex(registerMsg.sensorIndex);
558:
559:     if(asim == NULL)
560:     {

```

```

561:     printf("Spa1Manager: No matching ASIM found for SDMRegister \n");
562:     return;
563: }
564:
565: debug_f(1, "Spa1Manager: SDMRegister Received for ASIM %i \n", asim->getSensorIndex());
566:
567: asim->setRegState(RECVD_REG);
568: }
569:
570: /**
571:  * Used to handle an SDMID message received by the SPA-1 Manager
572:  *
573:  * When an SDMID message is received, this function sets the
574:  * corresponding ASIMs registration state to RECVD_ID. This causes
575:  * the ASIM's registration thread to finish as the ASIM has now
576:  * fully registered.
577:  *
578:  * @param buf The buffer containing the received SDMID msg
579:  *
580:  */
581: void handleIdMsg(char* buf)
582: {
583:     SDMID idMsg;
584:     Spa1Asim* asim;
585:     if(idMsg.Unmarshal(buf) < 0)
586:     {
587:         printf("Bad SDMID Msg received \n");
588:     }
589:
590:     asim = asimTable.getAsimBySensorId(idMsg.destination.getSensorID());
591:
592:     if(asim == NULL)
593:     {
594:         printf("No matching ASIM found for SDMID \n");
595:         return;
596:     }
597:
598:     debug_f(1, "Spa1Manager: SDMID Received for ASIM %i \n", asim->getSensorIndex());
599:
600:     asim->setRegState(RECVD_ID);
601:     asim->setSensorId(idMsg.destination.getSensorID());

```

```

602: }
603:
604: /**
605:  * Used to handle an SDMSerreqst message received by the SPA-1 Manager
606:  *
607:  * When an SDMSerreqst message is received, this function checks to see the
608:  * intended destination. If it is for an ASIM, the command is passed along to
609:  * the corresponding ASIM. If it is for the Spa1Manger, it is handled accordingly.
610:  *
611:  * @param buf The buffer containing the received SDMSerreqst msg
612:  *
613:  */
614: void handleSerreqstMsg(char* buf)
615: {
616:     SDMSerreqst serReqstMsg;
617:     Spa1Asim* asim;
618:     unsigned char msgBuf[MAX_SPA1_MSG_SIZE];
619:     int msgSize;
620:
621:     if (serReqstMsg.Unmarshal(buf) < 0)
622:     {
623:         printf("Spa1Manager: Invalid SDMSerreqst message \n");
624:         return;
625:     }
626:
627:     if((serReqstMsg.source.getSensorID() & 0x0ff) != 1) //Destined for an ASIM
628:     {
629:         debug_f(1,"Spa1Manager: Service request received for ASIM %hu \n",
serReqstMsg.source.getSensorID() & 0x00ff);
630:
631:         asim = asimTable.getAsimBySensorId(serReqstMsg.source.getSensorID());
632:         if(asim != NULL)
633:         {
634:             msgSize = translator.translateToSpa1((unsigned char*)buf, msgBuf,
MAX_SPA1_MSG_SIZE);
635:             if(asim->sendToQueue.putMsg(msgBuf, msgSize) == 0)
636:             {
637:                 debug_f(1, "Spa1Manager: Added Service Request Msg destined for ASIM %hu to
its queue \n", asim->getSensorIndex());
638:             }
639:             else
640:             {

```

```

641:         printf("Spa1Manager: Error adding request msg to ASIM queue \n");
642:     }
643: }
644: else
645: {
646:     debug_f(1, "Spa1Manager: No matching ASIM (%hu) found for SDMSerreqst Msg \n",
serReqstMsg.source.getSensorID() & 0x00ff);
647: }
648: }
649: else//Destined for Spa1Manager
650: {
651:     if(serReqstMsg.command_id == getxTEDSMsg)
652:     {
653:         getAsimxTEDSReply(buf);
654:     }
655:     else if(serReqstMsg.command_id == getAsimDetailsMsg)
656:     {
657:         getAsimDetailsReply(buf);
658:     }
659: }
660: }
661:
662: /**
663:  * Used to handle an a request made to the Spa1Manager for the xTEDS of a
664:  * particular SPA-1 ASIM
665:  *
666:  * @param buf The buffer containing the received SDMSerreqst msg
667:  *
668:  */
669: void getAsimxTEDSReply(char* buf)
670: {
671:     SDMSerreqst serReqstMsg;
672:     SDMData dataMsg;
673:     unsigned int sensorId;
674:     unsigned short xTEDSLen;
675:     Spa1Asim* asim;
676:
677:     serReqstMsg.Unmarshal(buf);
678:     dataMsg.source = serReqstMsg.source;
679:     dataMsg.msg_id = serReqstMsg.reply_id;
680:     sensorId = GET_UINT(serReqstMsg.data);

```

```

681:
682:     asim = asimTable.getAsimBySensorId(sensorId);
683:
684:     if(asim != NULL)
685:     {
686:         xTEDSLen = (unsigned short)asim->xTEDSSize;
687:         PUT_USHORT(dataMsg.msg, xTEDSLen);
688:         memcpy(&dataMsg.msg[2], asim->xTEDS, xTEDSLen);
689:         dataMsg.length = xTEDSLen + sizeof(unsigned short);
690:         dataMsg.Send(serReqstMsg.destination);
691:     }
692: }
693:
694: /**
695:  * Used to handle an a request made to the Spa1Manager for details pertaining to
696:  * a specific SPA-1 ASIM
697:  *
698:  * @param buf The buffer containing the received SDMSerreqst msg
699:  */
700: void getAsimDetailsReply(char* buf)
701: {
702:     SDMSerreqst serReqstMsg;
703:     SDMData dataMsg;
704:     unsigned int sensorId;
705:     Spa1Asim* asim;
706:     int cur = 0;
707:     char nameBuf[25];
708:     int nameSize;
709:
710:     serReqstMsg.Unmarshal(buf);
711:     dataMsg.source = serReqstMsg.source;
712:     dataMsg.msg_id = serReqstMsg.reply_id;
713:     sensorId = GET_UINT(serReqstMsg.data);
714:
715:     asim = asimTable.getAsimBySensorId(sensorId);
716:
717:     if(asim != NULL)
718:     {
719:         PUT_UINT(&dataMsg.msg[cur], sensorId);
720:         cur += 4;

```

```

722:     dataMsg.msg[cur] = asim->getI2cAddress();
723:     cur += 1;
724:     PUT_UINT(&dataMsg.msg[cur], sensorId);
725:     cur += 4;
726:     nameSize = asim->getAsimName(nameBuf);
727:     memcpy(&dataMsg.msg[cur], nameBuf, nameSize);
728:     cur += nameSize;
729:
730:     dataMsg.length = cur;
731:     dataMsg.Send(serReqstMsg.destination);
732: }
733: }
734:
735:
736: /**
737:  * Used to handle an SDMCommand message received by the SPA-1 Manager
738:  *
739:  * When an SDMCommand message is received, this function checks to see the
740:  * intended destination. If it is for an ASIM, the command is passed along to
741:  * the corresponding ASIM. If it is for the Spa1Manger, it is handled accordingly.
742:  *
743:  * @param buf The buffer containing the received SDMCommand msg
744:  *
745:  */
746: void handleCommandMsg(char* buf)
747: {
748:     SDMCommand commandMsg;
749:     Spa1Asim* asim;
750:     unsigned char msgBuf[MAX_SPA1_MSG_SIZE];
751:     int msgSize;
752:
753:     if (commandMsg.Unmarshal(buf) < 0)
754:     {
755:         printf("Spa1Manager: Invalid SDMCommand message \n");
756:         return;
757:     }
758:
759:     if((commandMsg.source.getSensorID() & 0x0ff) != 1) //Destined for an ASIM
760:     {
761:         debug_f(1,"Spa1Manager: Command Msg received for ASIM %hu \n",
commandMsg.source.getSensorID() & 0x00ff);

```

```

762:
763:     asim = asimTable.getAsimBySensorId(commandMsg.source.getSensorID());
764:     if(asim != NULL)
765:     {
766:         msgSize = translator.translateToSpa1((unsigned char*)buf, msgBuf,
MAX_SPA1_MSG_SIZE);
767:         if(asim->sendToQueue.putMsg(msgBuf, msgSize) == 0)
768:         {
769:             debug_f(1, "Spa1Manager: Added Command Msg destined for ASIM %hu to its
queue \n", asim->getSensorIndex());
770:         }
771:     } else
772:     {
773:         printf("Spa1Manager: Error adding command msg to ASIM queue \n");
774:     }
775: }
776: else
777: {
778:     debug_f(1, "Spa1Manager: No matching ASIM (%hu) found for SDMCommand Msg
\n", commandMsg.source.getSensorID() & 0x00ff);
779: }
780: }
781: else//Destined for Spa1Manager
782: {
783:     if(commandMsg.command_id == resetBusMsg)
784:     {
785:         //TODO: Send reset messages to each ASIM?
786:         //Perhaps close the i2c file descriptor and remove/re-insert the i2c kernel module?
787:         //We would have to basically suspend all operations of the SPA-1 Manager during this
global reset
788:     }
789: }
790: }
791:
792:
793: /**
794: * Used to handle an SDMSubreqst message received by the SPA-1 Manager
795: *
796: * When an SDMSubreqst message is received, this function checks to see the
797: * intended destination. If it is for an ASIM, the subscription is stored for the ASIM.
798: * If it is for the Spa1Manger, it is recoreded in the manager's subscription manager.
799: *

```

```

800: * @param buf The buffer containing the received SDMSubreqst msg
801: *
802: */
803: void handleSubreqstMsg(char* buf)
804: {
805:     SDMSubreqst subReqstMsg;
806:     Spa1Asim* asim;
807:     unsigned char msgBuf[MAX_SPA1_MSG_SIZE];
808:     int msgSize;
809:
810:     if (subReqstMsg.Unmarshal(buf) < 0)
811:     {
812:         printf("Spa1Manager: Invalid SDMSubreqst message \n");
813:         return;
814:     }
815:
816:     if((subReqstMsg.source.getSensorID() & 0x0ff) != 1) //Destined for an ASIM
817:     {
818:         debug_f(1,"Spa1Manager: Subreqst Msg received for ASIM %hu \n",
subReqstMsg.source.getSensorID() & 0x00ff);
819:
820:         asim = asimTable.getAsimBySensorId(subReqstMsg.source.getSensorID());
821:         if(asim != NULL)
822:         {
823:             msgSize = translator.translateToSpa1((unsigned char*)buf, msgBuf,
MAX_SPA1_MSG_SIZE);
824:             asim->subscriptions.AddSubscription(subReqstMsg);
825:             debug_f(1, "Spa1Manager: Subscription added for ASIM %hu msg(%i,%i) \n",
subReqstMsg.source.getSensorID() & 0x00ff, subReqstMsg.msg_id.getInterface(),
subReqstMsg.msg_id.getMessage());
826:             if(asim->sendToQueue.putMsg(msgBuf, msgSize) == 0)
827:             {
828:                 debug_f(1, "Spa1Manager: Added data request msg destined for ASIM %hu to its
queue \n", asim->getSensorIndex());
829:             }
830:             else
831:             {
832:                 printf("Spa1Manager: Error adding data request msg to ASIM queue \n");
833:             }
834:         }
835:     }
836:     {

```



```

837:         debug_f(1, "Spa1Manager: No matching ASIM (%hu) found for SDMSubreqst Msg \n",
subReqstMsg.source.getSensorID() & 0x00ff);
838:     }
839: }
840: else//Destined for Spa1Manager
841: {
842:     subscriptions.AddSubscription(subReqstMsg);
843: }
844: }
845:
846: /**
847: * Used to handle an SDMDeletesub message received by the SPA-1 Manager
848: *
849: * When an SDMDeletesub message is received, this function checks to see the
850: * intended destination. If it is for an ASIM, the subscription is removed for the ASIM.
851: * If it is for the Spa1Manger, it is removed from the manager's subscription manager.
852: *
853: * @param buf The buffer containing the received SDMDeletesub msg
854: *
855: */
856: void handleDeletesubMsg(char* buf)
857: {
858:     SDMDeletesub delSubMsg;
859:     Spa1Asim* asim;
860:
861:     if (delSubMsg.Unmarshal(buf) < 0)
862:     {
863:         printf("Spa1Manager: Invalid SDMSubreqst message \n");
864:         return;
865:     }
866:
867:     if((delSubMsg.source.getSensorID() & 0x0ff) != 1) //Destined for an ASIM
868:     {
869:         debug_f(1,"Spa1Manager: Deletesub Msg received for ASIM %hu \n",
delSubMsg.source.getSensorID() & 0x00ff);
870:
871:         asim = asimTable.getAsimBySensorId(delSubMsg.source.getSensorID());
872:         if(asim != NULL)
873:         {
874:             asim->subscriptions.RemoveSubscription(delSubMsg);

```

```

875:         debug_f(1, "Spa1Manager: Subscription removed for ASIM %hu msg(%i,%i) \n",
delSubMsg.source.getSensorID()      &      0x00ff,      delSubMsg.msg_id.getInterface(),
delSubMsg.msg_id.getMessage());
876:     }
877:     else
878:     {
879:         debug_f(1, "Spa1Manager: No matching ASIM (%hu) found for SDMDelatesub Msg \n",
delSubMsg.source.getSensorID() & 0x00ff);
880:     }
881: }
882: else//Destined for Spa1Manager
883: {
884:     subscriptions.RemoveSubscription(delSubMsg);
885: }
886: }
887:
888:
889: /**
890: * Used for registering a SPA-1 ASIMs xTEDS with the SDM
891: *
892: * This function should only be called after an SDMRegister message
893: * is received for a specific ASIM. The xTEDS is sent and the Spa1Asim's
894: * registration state is updated to SENT_XTEDS.
895: *
896: * @param buf The buffer containing the received SDMID msg
897: *
898: */
899: void registerAsimXteds(Spa1Asim* asim)
900: {
901:     SDMxTEDS xTEDSMsg;
902:     xTEDSMsg.source.setSensorID(asim->getSensorIndex());
903:     xTEDSMsg.source.setPort(PORT_SPA1_MANAGER);
904:     strcpy(xTEDSMsg.xTEDS, asim->xTEDS);
905:     xTEDSMsg.Send();
906:     debug_f(1, "Spa1Manager: SDMxTEDS sent for ASIM %i \n", asim->getSensorIndex());
907:     asim->setRegState(SENT_XTEDS);
908: }
909:
910: /**
911: * Used for canceling both the Spa1Manager's xTEDS with the SDM, as well as the
912: * xTEDS of all registered SPA-1 ASIMs.
913: *

```

```

914: */
915: void cancelSdmRegistrations()
916: {
917:     SDMCancelxTEDS cancelxTEDSMsg;
918:
919:     asimTable.cancelSdmRegistrations();
920:
921:     cancelxTEDSMsg.source.setPort(PORT_SPA1_MANAGER);
922:     cancelxTEDSMsg.source.setSensorID(1);    //Cancel the Spa1Manager's xTEDS
923:     cancelxTEDSMsg.Send();
924: }
925:
926: /**
927:  * Linux signal handler to catch SIGINT and gracefully shut down the Spa1Manager and
928:  * deregister itself and all ASIMs with the SDM.
929:  *
930:  */
931: #ifndef WIN32
932: void* signalHandler(void* args) //signal handler for linux
933: {
934:     sigset_t signal_set;
935:     int sig;
936:     sigemptyset(&signal_set);
937:     sigaddset(&signal_set, SIGINT);
938:
939:     while (!termination)
940:     {
941:         sigwait(&signal_set, &sig);
942:         switch(sig)
943:         {
944:             case SIGINT:
945:                 printf("Spa1Manager: Shutting Down \n");
946:                 termination = true;
947:                 cancelSdmRegistrations();
948:                 break;
949:         }
950:     }
951:     return NULL;
952: }
953: #endif
954:

```

```

955:
956: #ifdef WIN32
957: /**
958: * Windows signal handler to catch SIGINT and gracefully shut down the Spa1Manager and
959: * deregister itself and all ASIMs with the SDM.
960: *
961: * @param sig The signal received
962: *
963: */
964: void signalHandler(int sig)
965: {
966:     switch (sig)
967:     {
968:         case SIGINT:
969:             printf("Spa1Manager: Shutting Down \n");
970:             termination = true;
971:             cancelSdmRegistrations();
972:             break;
973:         default:
974:             break;
975:     }
976: }
977: #endif
978:
979: /**
980: * Utility function for retrieving the system time in seconds.
981: * Useful for simple polling timers.
982: *
983: * @return An int containing the current system clock value of seconds
984: *
985: */
986: unsigned int getCurTime()
987: {
988:     unsigned int seconds;
989:     unsigned int uSeconds;
990:     SDM_GetTime(&seconds, &uSeconds);
991:
992:     return seconds;
993: }

```

File: sdm/Spa1Manager/Spa1Asim.h

```
1: #ifndef __SPA1_ASIM_H__
2: #define __SPA1_ASIM_H__
3:
4: #include "../common/SubscriptionManager/SubscriptionManager.h"
5: #include "../common/sdmLib.h"
6: #include "../common/message_defs.h"
7: #include "../common/Time/SDMTime.h"
8: #include "../common/Debug.h"
9: #include "Spa1Queue.h"
10: #include <unistd.h>
11: #include <stdio.h>
12: #include <fcntl.h>
13: #include <sys/ioctl.h>
14: #ifndef WIN32
15: #include <linux/i2c.h> /* for I2C_SLAVE */
16: #endif
17: #ifndef I2C_SLAVE
18: #define I2C_SLAVE 0x0703 /* Change slave address */
19: #endif
20:
21:
22: #define SPA1_HEADER_SIZE 3
23: #define MAX_PACKET_PAYLOAD 253
24: #define MAX_ASIM_MSG_SIZE 256
25: #define ASIM_READ_TIMEOUT 3
26: #define MAX_ASIM_FAILURES 3
27:
28: enum RegistrationState { SENT_HELLO, RECVD_ACK, RECVD_REG, SENT_XTEDS,
RECVD_ID };
29:
30: /**
31: * @class Spa1Asim
32: *
33: * @brief Defines the Spa1Asim object for use by the SPA-1 Sensor Manager
34: *
35: * The Spa1Asim objects holds all information about a single SPA-1 ASIM that
36: * is registered with the SPA-1 Manager. Utility functions are also included
37: * to write to/read from the ASIM.
38: *
```

```

39: * @author Bryan Hansen, Jacob Christensen
40: * @date 12/03/2009
41: */
42: class Spa1Asim
43: {
44:     public:
45:         char* xTEDS; //Holds this ASIM's xTEDS in plain text format
46:         int xTEDSSize;
47:
48:         ///Queue to send messages to the ASIM
49:         Spa1Queue sendToQueue;
50:
51:         ///Queue used to recv messages from the ASIM
52:         Spa1Queue recvFromQueue;
53:
54:         ///Manages subscriptions for this ASIM
55:         SubscriptionManager subscriptions;
56:
57:         ///Pointer to next Spa1Asim in a list
58:         Spa1Asim* next;
59:
60:         Spa1Asim(int busFd, unsigned char newAddress, int guid);
61:         int recvFromAsim(unsigned char* buf, size_t bufSize, bool blocking = false);
62:         int recvXtedsFromAsim();
63:         int sendToAsim(unsigned char* buf, size_t msgSize);
64:         bool doRegistration(void);
65:
66:
67:         //Getters/Setters for private members
68:         unsigned char getI2cAddress();
69:         void setI2CAddress(unsigned char newAddress);
70:         unsigned int getSensorIndex();
71:         void setSensorIndex(unsigned short newSensorIndex);
72:         unsigned int getSensorId();
73:         void setSensorId(unsigned long newSensorId);
74:         RegistrationState getRegState();
75:         void setRegState(RegistrationState newState);
76:         int getGuid();
77:         void incrementFailCount();
78:         int getFailCount();
79:         int getAsimName(char* outBuf);

```

```

80:
81: private:
82:     ///File descriptor used for reads/writes to i2c bus
83:     int i2cFd;
84:
85:     /// Used to uniquely identify an ASIM during registration
86:     int guid;
87:
88:     ///ASIMs address on I2C bus
89:     unsigned char i2cAddress;
90:
91:     ///SPA-1 Manager sensor Index, also lower two bytes of SDM SensorID
92:     unsigned int sensorIndex;
93:
94:     ///SDM sensor id, received in SDMID msg after successful registration
95:     unsigned int sensorId;
96:
97:     ///Registration state of this ASIM
98:     RegistrationState regState;
99:
100:    ///Number of times a nack was received when attempting a read from the ASIM
101:    int failCount;
102:
103:    ///Private utility functions
104:    int readPacketHeader(unsigned char* buf, bool doBlocking = false);
105:    double getCurTime();
106: };
107:
108: #endif

```

File: sdm/Spa1Manager/Spa1Asim.cpp

```
1: #include "Spa1Asim.h"
2:
3: extern int debug;
4:
5: /**
6:  * Constructor that creates a Spa1ASIM object
7:  */
8: Spa1Asim::Spa1Asim(int busFd, unsigned char newAddress, int guid) : i2cFd(busFd),
i2cAddress(newAddress), guid(guid), xTEDSSize(0), failCount(0), next(NULL)
9: {
10:     // do nothing
11: }
12:
13: /**
14:  * Utility function for receiving messages from the ASIM.
15:  *
16:  * @param buf The output buffer the message will be written into
17:  * @param bufSize The size of the output buffer
18:  *
19:  * @return An int containing the number of bytes actually read, -1 indicates
20:  *         an error condition has occurred.
21:  */
22: int Spa1Asim::recvFromAsim(unsigned char* buf, size_t bufSize, bool blocking)
23: {
24:
25: #ifndef WIN32
26:     if (ioctl(i2cFd, I2C_SLAVE, i2cAddress) < 0)
27:     {
28:         printf("SPA-1 Manager: Error setting i2c address in recv \n");
29:     }
30: #endif
31:
32:     short length = 0;
33:     int bytesRead = 0;
34:
35:     bytesRead = readPacketHeader(buf, blocking);
36:
37:     if (bytesRead == 0)
38:     {
```



```

39:     return 0;
40: }
41:
42: if (bytesRead == -1)
43: {
44:     return bytesRead;
45: }
46:
47: memcpy(&length, &buf[1], 2);
48:
49:     if ((unsigned short)(length + SPA1_HEADER_SIZE) > bufSize || length >
MAX_PACKET_PAYLOAD)
50: {
51:     return -1;
52: }
53:
54: bytesRead += read(i2cFd, &buf[3], length);
55:
56: return bytesRead;
57: }
58:
59: /**
60: * Utility function for receiving messages from the ASIM in a blocking manner.
61: *
62: * @param buf The output buffer the message will be written into
63: * @param bufSize The size of the output buffer
64: *
65: * @return An int containing the number of bytes actually read, -1 indicates
66: *         an error condition has occurred.
67: */
68: int Spa1Asim::recvXtedsFromAsim()
69: {
70: #ifndef WIN32
71:     if (ioctl(i2cFd, I2C_SLAVE, i2cAddress) < 0)
72:     {
73:         printf("SPA-1 Manager: Error setting i2c address in recv \n");
74:     }
75: #endif
76:
77:     short length = 0;
78:     int bytesRead = 0;

```

```

79: unsigned char headerBuf[3];
80:
81: usleep(1000);
82: readPacketHeader(headerBuf, true);
83:
84: memcpy(&length, &headerBuf[1], 2);
85:
86: debug_f(4, "Read xTEDS Header: 0x%02x w/ length: %i \n", headerBuf[0], length);
87:
88: xTEDS = new char[length + 1];
89: memset(xTEDS, 0, length + 1);
90:
91: usleep(1000);
92:
93: if (length <= MAX_PACKET_PAYLOAD)
94: {
95:     bytesRead = read(i2cFd, &xTEDS[3], length);
96: }
97: else
98: {
99:     // loop and reassemble packets
100:     int index = 0;
101:     int numberPackets = length / MAX_PACKET_PAYLOAD;
102:     int lastPacketLength = length % MAX_PACKET_PAYLOAD;
103:
104:     // read the rest of the first packet
105:     bytesRead = read(i2cFd, &xTEDS[index], MAX_PACKET_PAYLOAD);
106:     debug_f(4, "Read %i bytes of xTEDS \n", bytesRead);
107:     index += MAX_PACKET_PAYLOAD;
108:
109:     // read all the other packets
110:     for (int packet = 1; packet < numberPackets; ++packet)
111:     {
112:         usleep(1000);
113:         readPacketHeader(headerBuf, true);
114:         debug_f(4, "Read xTEDS Header: 0x%02x 0x%02x 0x%02x \n", headerBuf[0],
headerBuf[1], headerBuf[2]);
115:         usleep(1000);
116:         bytesRead += read(i2cFd, &xTEDS[index], MAX_PACKET_PAYLOAD);
117:         debug_f(4, "Read %i bytes of xTEDS \n", bytesRead);
118:         index += MAX_PACKET_PAYLOAD;

```

```

119:     }
120:
121:     // read the last packet
122:     if (lastPacketLength > 0)
123:     {
124:         usleep(1000);
125:         readPacketHeader(headerBuf, true);
126:         debug_f(4, "Read xTEDS Header: 0x%02x 0x%02x 0x%02x \n", headerBuf[0],
headerBuf[1], headerBuf[2]);
127:         usleep(1000);
128:         bytesRead += read (i2cFd, &xTEDS[index], lastPacketLength);
129:         debug_f(4, "Read %i bytes of xTEDS \n", bytesRead);
130:     }
131:
132: }
133:
134: return bytesRead;
135: }
136:
137: /**
138:  * Utility function for reading just the header from an ASIM msg
139:  *
140:  * @param buf The buffer to read the header into
141:  * @param doBlocking Whether this function should continue attempting to read until
142:  *         data becomes available.
143:  *
144:  * @return An int containing the number of bytes actually read, -1 indicates
145:  *         an unresponsive ASIM. 0 indicates that the ASIM responded, but did
146:  *         not have any data available for reading. If doBlocking is true, -1
147:  *         can also signify a timeout.
148:  */
149: int Spa1Asim::readPacketHeader(unsigned char* buf, bool doBlocking)
150: {
151:     int bytesRead = 0;
152:
153:     if (!doBlocking)
154:     {
155:         bytesRead = read(i2cFd, buf, SPA1_HEADER_SIZE);
156:
157:         if (bytesRead == -1)
158:         {

```

```

159:     return bytesRead;
160: }
161:
162: if (buf[0] == 0xff && buf[1] == 0xff && buf[2] == 0xff)
163: {
164:     bytesRead = 0;
165: }
166: }
167: else
168: {
169:     bool dataReady = false;
170:     double endTime = getCurTime() + ASIM_READ_TIMEOUT;
171:
172:     while (!dataReady)
173:     {
174:         bytesRead = read(i2cFd, buf, SPA1_HEADER_SIZE);
175:
176:         if (bytesRead == -1)
177:         {
178:             break;
179:         }
180:
181:         if (buf[0] == 0xff && buf[1] == 0xff && buf[2] == 0xff)
182:         {
183:             dataReady = false;
184:             usleep(1000);
185:         }
186:         else
187:         {
188:             dataReady = true;
189:         }
190:
191:         if (getCurTime() > endTime)
192:         {
193:             bytesRead = -1;
194:             break;
195:         }
196:     }
197: }
198:
199: return bytesRead;

```

```

200: }
201:
202:
203:
204:
205: /**
206:  * Utility function for sending messages to the ASIM.
207:  *
208:  * @param buf The buffer containing the message to be sent
209:  * @param msgSize The size of the message to send in bytes
210:  *
211:  * @return An int containing the number of bytes actually written, -1 indicates
212:  *         an error condition has occurred.
213:  */
214: int Spa1Asim::sendToAsim(unsigned char* buf, size_t msgSize)
215: {
216: #ifndef WIN32
217:   if (ioctl(i2cFd, I2C_SLAVE, i2cAddress) < 0)
218:   {
219:     printf("SPA-1 Manager: Error setting i2c address in send \n");
220:   }
221: #endif
222:
223:   return write(i2cFd, buf, msgSize);
224: }
225:
226: /**
227:  * Returns the current i2c address of the ASIM
228:  *
229:  * @return An unsigned char containing the I2C address
230:  */
231: unsigned char Spa1Asim::getI2cAddress()
232: {
233:   return i2cAddress;
234: }
235:
236: /**
237:  * Sets the 1-byte I2C address of the ASIM
238:  *
239:  * @param newAddress The address to set
240:  */

```

```

241: void Spa1Asim::setI2CAddress(unsigned char newAddress)
242: {
243:     i2cAddress = newAddress;
244: }
245:
246: /**
247:  * Returns the current sensor index of the ASIM
248:  *
249:  * @return An unsigned int containing the ASIM's sensor index
250:  */
251: unsigned int Spa1Asim::getSensorIndex()
252: {
253:     return sensorIndex;
254: }
255:
256: /**
257:  * Sets the sensor indexd of this ASIM
258:  *
259:  * @param newSensorId The sensor Index to set
260:  */
261: void Spa1Asim::setSensorIndex(unsigned short newSensorIndex)
262: {
263:     sensorIndex = newSensorIndex;
264: }
265:
266: /**
267:  * Returns the current sensorId of the ASIM
268:  *
269:  * @return An unsigned int containing the ASIM's sensor id
270:  */
271: unsigned int Spa1Asim::getSensorId()
272: {
273:     return sensorId;
274: }
275:
276: /**
277:  * Sets the sensor id of this ASIM
278:  *
279:  * @param newSensorId The sensor Id to set
280:  */
281: void Spa1Asim::setSensorId(unsigned long newSensorId)

```

```

282: {
283:   sensorId = newSensorId;
284: }
285:
286: /**
287:  * Returns the current registration state
288:  *
289:  * @return A RegistrationState enum indicating this ASIM current registration state
290:  */
291: RegistrationState Spa1Asim::getRegState()
292: {
293:   return regState;
294: }
295:
296: /**
297:  * Sets the registration state of the ASIM
298:  *
299:  * @param newState The registration state to set
300:  */
301: void Spa1Asim::setRegState(RegistrationState newState)
302: {
303:   regState = newState;
304: }
305:
306: /**
307:  * Performs the registration of the ASIM with the SM
308:  */
309: bool Spa1Asim::doRegistration(void)
310: {
311:   unsigned char outBuf[16];
312:   unsigned char inBuf[BUFSIZE];
313:
314:   memset(outBuf, 0, 16);
315:   outBuf[0] = 'I';
316:   outBuf[1] = 0;
317:   outBuf[2] = 0;
318:
319:   this->sendToAsim(outBuf, 3);      // Send I
320:   debug_f(4, "Sent (I) msg \n");
321:   usleep(1000);
322:   int bytesRead = this->recvFromAsim(inBuf, BUFSIZE, true); // Get S blocking!

```

```

323: //printf ("After I bytesRead == %i \n", bytesRead);
324:  debug_f(4, "Received (S) \n");
325:  if(debug >= 4)
326:  {
327:      for(int i = 0; i < bytesRead; i++)
328:      {
329:          debug_f(4, "buf[%i]: 0x%02x \n", i, inBuf[i]);
330:      }
331:  }
332:
333: //printf("Got S: %x \n", inBuf[3]);
334: if (bytesRead == -1)
335: {
336:     return false;
337: }
338:
339: // See the page about the status message at:
340: // https://gonzales.cs.usu.edu/drupal/node/47
341: if ((inBuf[3] & 0xe0) != 0)
342: {
343:     return false;
344: }
345:
346: outBuf[0] = 'U';
347: this->sendToAsim(outBuf, 3);      // Send U
348:  debug_f(4, "Sent (U) msg \n");
349:  usleep(1000);
350: this->recvFromAsim(inBuf, BUFSIZE, true); // Get K
351:  debug_f(4, "Received (K) \n");
352:  if(debug >= 4)
353:  {
354:      for(int i = 0; i < 4; i++)
355:      {
356:          debug_f(4, "buf[%i]: 0x%02x \n", i, inBuf[i]);
357:      }
358:  }
359: //printf("Got K: %x \n", inBuf[3]);
360: // there is no invalid version and it means nothing at this point
361:
362: outBuf[0] = 'X';
363: this->sendToAsim(outBuf, 3);      // Send X

```



```

364:   debug_f(4, "Sent (X) msg \n");
365:   usleep(1000);
366:   xTEDSSize = this->recvXtedsFromAsim();
367:   debug_f(4, "Received (J) size: %i \n", xTEDSSize);
368:
369:   if (debug >= 4)
370:   {
371:       printf ("Got xTEDS: ");
372:       for (int i = 0; i < xTEDSSize; ++i)
373:       {
374:           printf("%c", xTEDS[i]);
375:       }
376:       printf (" \n");
377:   }
378:
379:   return true;
380: }
381:
382: /**
383:  * Increments the number of failures associated with this ASIM
384:  *
385:  */
386: void Spa1Asim::incrementFailCount()
387: {
388:     failCount++;
389: }
390:
391: /**
392:  * Returns the current number of failures for this ASIM
393:  *
394:  * @return An int containing the current failure count
395:  */
396: int Spa1Asim::getFailCount()
397: {
398:     return failCount;
399: }
400:
401:
402: /**
403:  * Private utility function used to return the system time
404:  *

```

```

405: * @return A double containing the current system time (seconds + partial seconds)
406: */
407: double Spa1Asim::getCurTime()
408: {
409:     unsigned int seconds;
410:     unsigned int uSeconds;
411:     double curTime;
412:     SDM_GetTime(&seconds, &uSeconds);
413:     curTime = seconds + ((double)uSeconds/1000000.0);
414:     return curTime;
415: }
416:
417:
418: /**
419: * Returns the guid used by this ASIM for arbitration
420: *
421: * @return An int containing guid used by the ASIM
422: */
423: int Spa1Asim::getGuid()
424: {
425:     return guid;
426: }
427:
428:
429: /**
430: * Returns the name (null terminated) of the ASIM as pulled from its xTEDS
431: *
432: * @param outBuf An output buffer which will have the ASIMs name place into
433: *               it. This buffer should be at least 25 bytes long.
434: *
435: * @return An int containing length of the null terminated ASIM name
436: */
437: int Spa1Asim::getAsimName(char* outBuf)
438: {
439:     char* temp;
440:     int nameSize;
441:     if(xTEDS != NULL)
442:     {
443:         temp = strstr(xTEDS, "Device name= \\");
444:
445:         for(nameSize = 0; nameSize < 24; nameSize++)

```

```

446:     {
447:         if(temp[nameSize + 13] == "")
448:         {
449:             memcpy(outBuf, &temp[13], nameSize);
450:             outBuf[nameSize] = '\0'; //null terminate
451:             break;
452:         }
453:     }
454:
455:     if(nameSize == 24)
456:     {
457:         memcpy(outBuf, &temp[13], nameSize);
458:         outBuf[nameSize] = '\0'; //null terminate
459:     }
460:
461:     return nameSize + 1; //return size of name + 1 for null terminator
462: }
463: else
464: {
465:     return 0; //No name available as xTEDS not yet registered
466: }
467: }
468:

```

File: sdm/Spa1Manager/Spa1Manager.h

```
1: #ifndef __SPA1_MANAGER_H__
2: #define __SPA1_MANAGER_H__
3:
4: #include <stdio.h>
5: #include <fcntl.h>
6: #include <unistd.h>
7: #include <string.h>
8: #include <pthread.h>
9: #include <sys/ioctl.h>
10: #include <sys/types.h>
11: #include <sys/stat.h>
12: #include <getopt.h>
13: #include <signal.h>
14: #include <stdlib.h>
15: #include "Spa1ManagerxTEDS.h"
16: #include "Spa1Asim.h"
17: #include "Spa1AsimTable.h"
18: #include "Spa1Translator.h"
19: #include "Spa1Queue.h"
20: #include "../common/MessageManager/MessageManager.h"
21: #include "../common/message/SDMHello.h"
22: #include "../common/message/SDMID.h"
23: #include "../common/message/SDMAck.h"
24: #include "../common/message/SDMRegister.h"
25: #include "../common/message/SDMxTEDS.h"
26: #include "../common/message/SDMCancelxTEDS.h"
27: #include "../common/message/SDMSerreqst.h"
28: #include "../common/message/SDMData.h"
29: #include "../common/message/SDMCommand.h"
30: #include "../common/Time/SDMTime.h"
31: #include "../common/Debug.h"
32: #include "../common/version.h"
33: #include "../common/SubscriptionManager/SubscriptionManager.h"
34:
35: #ifndef WIN32
36: #include <linux/i2c.h> /* for I2C_SLAVE */
37: #endif
38: #ifndef I2C_SLAVE
39: #define I2C_SLAVE 0x0703 /* Change slave address */
```

```

40: #endif
41:
42: ///Defines
43: #define THREAD_STACK_SIZE 256000
44: #define MAX_SPA1_ASIMS 64
45: #define DEFAULT_ADDRESS 0x11
46: #define DEFAULT_BUFFER_SIZE 16
47: #define I2C_READ_TIMEOUT 3
48: #define MAX_SPA1_MSG_SIZE 256
49:
50:
51: ///Function Prototypes
52: void* sdmListener(void* args);
53: void* i2cComm(void* args);
54: void* regAsimWithSdm(void* args);
55: #ifndef WIN32
56: void* signalHandler(void* args);
57: #else
58: void signalHandler(int sig);
59: #endif
60:
61: void asimMonitor();
62: int sdmRegister(MessageManager* msgManager);
63: void registerxTEDS();
64: void registerAsimXteds(Spa1Asim* asim);
65: void cancelSdmRegistrations();
66: unsigned char searchForNewAsims(unsigned char);
67: int i2cRead(unsigned char* buf, size_t size);
68: void setSlaveAddress(unsigned char address);
69: unsigned int getCurTime();
70:
71: void handleAckMsg(char* buf);
72: void handleRegisterMsg(char* buf);
73: void handleIdMsg(char* buf);
74: void handleSerreqstMsg(char* buf);
75: void handleCommandMsg(char* buf);
76: void sendCommandToAsim(Spa1Asim* asim, SDMCommand* commandMsg);
77: void handleSubreqstMsg(char* buf);
78: void handleDeletesubMsg(char* buf);
79: void getAsimxTEDSReply(char* buf);
80: void getAsimDetailsReply(char* buf);

```

```
81:
82:
83: ///Globals
84: pthread_t sdmListenThread;
85: pthread_t i2cThread;
86: pthread_t sigHandler;
87: Spa1AsimTable asimTable;
88: Spa1Translator translator;
89: SubscriptionManager subscriptions;
90: int busFd;
91: int debug = 0;
92: bool termination = false;
93:
94: SDMMMessage_ID regChangeMsg(1,1);
95: SDMMMessage_ID getXTEDSMsg(1,2);
96: SDMMMessage_ID xTEDSReplyMsg(1,3);
97: SDMMMessage_ID getAsimDetailsMsg(1,4);
98: SDMMMessage_ID asimDetailsReplyMsg(1,5);
99: SDMMMessage_ID resetBusMsg(2,1);
100:
101: #endif
```

File: sdm/Spa1Manager/Spa1ManagerxTEDS.h

```
1: #ifndef __SPA1_XTEDS_H__
2: #define __SPA1_XTEDS_H__
3:
4: const char* spa1ManagerxTEDS = "<?xml version= \"1.0\" encoding= \"utf-8\" ?> \n \
5:   <xTEDS xmlns= \"http://www.interfacecontrol.com/SPA/xTEDS \" xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \" \n \
6:   xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"
name= \"Spa1Manager \" version= \"1.0\"> \n \
7:   <Application name= \"Spa1Manager \" kind= \"Software \" description= \"SDM Core component
used to manage a SPA-1 network \" /> \n \
8:   <Interface name= \"AsimInfo \" id= \"1 \" description= \"Data relating to the SPA-1 ASIMs
registered with the Spa1Manager \"> \n \
9:     <Variable name= \"numRegistered \" kind= \"counter \" format= \"UINT32 \" description=
\"Number of SPA-1 ASIMs currently registered \" /> \n \
10:    <Variable name= \"sensorID \" kind= \"ID \" format= \"UINT32 \" description= \"The sensorID
of a SPA-1 ASIM \" /> \n \
11:    <Variable name= \"name \" kind= \"string \" format= \"INT08 \" length= \"25 \" /> \n \
12:    <Variable name= \"i2cAddress \" kind= \"index \" format= \"UINT08 \" description= \"The I2C
address an ASIM is currently registered on \" /> \n \
13:    <Variable name= \"guid \" kind= \"ID \" format= \"UINT32 \" description= \"GUID used by an
ASIM for I2C arbitration \" /> \n \
14:    <Variable name= \"regType \" kind= \"status \" format= \"UINT08 \" description= \"0 for
registrations, 1 for deregistrations \" /> \n \
15:    <Variable name= \"xTEDS \" kind= \"string \" format= \"INT08 \" /> \n \
16:    <Variable name= \"xTEDSSize \" kind= \"counter \" format= \"UINT16 \" /> \n \
17:    <Notification> \n \
18:    <DataMsg name= \"RegistrationChange \" description= \"Change in ASIM registrations \" id=
\"1 \" msgArrival= \"EVENT \"> \n \
19:      <VariableRef name= \"regType \" /> \n \
20:      <VariableRef name= \"sensorID \" /> \n \
21:    </DataMsg> \n \
22:  </Interface> \n \
23:  <Request> \n \
24:    <CommandMsg name= \"xTEDSById \" description= \"Get an xTEDS for an ASIM by its
SensorID \" id= \"2 \"> \n \
25:      <VariableRef name= \"sensorID \" /> \n \
26:    </CommandMsg> \n \
27:    <DataReplyMsg name= \"xTEDSReply \" description= \"Contains the requested xTEDS \" id=
\"3 \"> \n \
28:      <VariableRef name= \"xTEDSSize \" /> \n \
29:      <VariableRef name= \"xTEDS \" /> \n \
```

```

30: \t \t \t</DataReplyMsg> \n \
31: \t \t</Request> \n \
32: \t \t<Request> \n \
33: \t \t \t<CommandMsg name= \"GetAsimDetails \" description= \"Retreive details about a registered
SPA-1 ASIM \" id= \"4 \" > \n \
34: \t \t \t \t<VariableRef name= \"sensorID \" /> \n \
35: \t \t \t</CommandMsg> \n \
36: \t \t \t<DataReplyMsg name= \"AsimDetails \" id= \"5 \" > \n \
37: \t \t \t \t<VariableRef name= \"sensorID \" /> \n \
38: \t \t \t \t<VariableRef name= \"i2cAddress \" /> \n \
39: \t \t \t \t<VariableRef name= \"guid \" /> \n \
40: \t \t \t \t<VariableRef name= \"name \" /> \n \
41: \t \t \t</DataReplyMsg> \n \
42: \t \t</Request> \n \
43: \t</Interface> \n \
44: \t<Interface name= \"Bus \" id= \"2 \" description= \"Data and controls pertaining to the SPA-1 Bus \"
> \n \
45: \t \t<Command> \n \
46: \t \t \t<CommandMsg name= \"ResetBus \" id= \"1 \" /> \n \
47: \t \t</Command> \n \
48: \t</Interface> \n \
49: </xTEDS>;
50:
51: #endif

```


Listing from directory: sdm/tm

File: sdm/tm/TmXtedsDefs.h

```
1: #ifndef SDM_TM_XTEDS_DEFS_H_
2: #define SDM_TM_XTEDS_DEFS_H_
3:
4: #include "../common/message/SDMMessage_ID.h"
5: #include "../common/message_defs.h"
6:
7: //
8: // xTEDS definitions
9:
10: // Message ids
11: const SDMMessage_ID NOTI_TM_STATUS(1,1);
12: const SDMMessage_ID NOTI_TASK_QUEUED(1,2);
13: const SDMMessage_ID NOTI_TASK_STARTED(1,3);
14: const SDMMessage_ID NOTI_TASK_FINISHED(1,4);
15: const SDMMessage_ID CMD_CHANGE_MODE(1,5);
16: const SDMMessage_ID CMD_START_TASK(1, 8);
17: const SDMMessage_ID CMD_KILL_TASK(1, 9);
18: const SDMMessage_ID SER_NAME_TO_PID(1,6);
19: const SDMMessage_ID SER_NAME_TO_PID_REPLY(1,7);
20: const SDMMessage_ID SER_GET_TASK_LIST(1, 10);
21: const SDMMessage_ID SER_TASK_LIST_REPLY(1, 11);
22: const SDMMessage_ID SER_GET_RUNNING_TASK_LIST(1, 15);
23: const SDMMessage_ID SER_RUNNING_TASK_LIST_REPLY(1, 16);
24: const SDMMessage_ID SER_GET_TASK_INFO(1, 12);
25: const SDMMessage_ID SER_TASK_INFO_REPLY(1, 13);
26: const SDMMessage_ID NOTI_PERF_COUNTERS(2,13);
27: const SDMMessage_ID CMD_ENABLE_LOGGING(3,16);
28: const SDMMessage_ID CMD_DISABLE_LOGGING(3,17);
29:
30: enum XtedsChangeModeStatusEnum
31: {
32:     MODE_STATUS_SOFT_RESET = 1,
33:     MODE_STATUS_HARD_RESET = 2
34: };
35: enum XtedsExecutionModeEnum
36: {
37:     EXECUTION_MODE_NORMAL = 1,
```

```

38: EXECUTION_MODE_ALWAYS_RUNNING = 2
39: };
40: enum XtedsArchitectureTypeEnum
41: {
42:     ARCH_TYPE_INTEL = 1,
43:     ARCH_TYPE_MICROBLAZE = 2,
44:     ARCH_TYPE_SPARC = 3
45: };
46: enum XtedsOsTypeEnum
47: {
48:     OS_TYPE_LINUX26 = 1,
49:     OS_TYPE_WIN32 = 2,
50:     OS_TYPE_VXWORKS = 3
51: };
52: enum XtedsMemTypeEnum
53: {
54:     MEM_TYPE_64 = 1,
55:     MEM_TYPE_128 = 2,
56:     MEM_TYPE_256 = 3,
57:     MEM_TYPE_512 = 4
58: };
59: enum XtedsTaskStateEnum
60: {
61:     TASK_STATE_QUEUED = 1,
62:     TASK_STATE_SCHEDULED = 2,
63:     TASK_STATE_RUNNING = 3,
64:     TASK_STATE_STOPPED = 4
65: };
66: enum XtedsTaskRetrievalLocation
67: {
68:     TASK_LOCATION_PRIMARY = 1,
69:     TASK_LOCATION_TEMPORARY = 2,
70:     TASK_LOCATION_BACKUP = 3
71: };
72:
73: const unsigned int XTEDS_MAX_TASK_NAME_SIZE = MAX_FILENAME_SIZE;
74:
75: /*xTEDS for the TaskManager*/
76: #define STR_TASK_MANAGER_XTEDS "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n \

```

```

77: <xTEDS version= \"2.0 \" xmlns= \"http://www.interfacecontrol.com/SPA/xTEDS \" xmlns:xsi=
\"http://www.w3.org/2001/XMLSchema-instance \" name= \"Task_Manager_xTEDS \"
xsi:schemaLocation= \"http://www.interfacecontrol.com/SPA/xTEDS ../Schema/xTEDS02.xsd \"/> \n \
78: \t<Application kind= \"Software \" name= \"TaskManager \"/> \n \
79: \t<Interface name= \"TM_Interface \" id= \"1 \"> \n \
80: \t \t<Variable format= \"UINT08 \" kind= \"TBD \" name= \"Mode \"/> \n \
81: \t \t \t<Drange name= \"TmModeEnum \"> \n \
82: \t \t \t \t<Option value= \"1 \" name= \"SoftReset \"/> \n \
83: \t \t \t \t<Option value= \"2 \" name= \"HardReset \"/> \n \
84: \t \t \t</Drange> \n \
85: \t \t</Variable> \n \
86: \t \t<Variable format= \"UINT08 \" kind= \"string \" name= \"TaskName \" length= \"128 \"/> \n \
87: \t \t<Variable format= \"INT32 \" kind= \"TBD \" name= \"ExitStatus \"/> \n \
88: \t \t<Variable format= \"UINT32 \" kind= \"IP_long \" name= \"SenderAddr \"/> \n \
89: \t \t<Variable format= \"UINT16 \" kind= \" Port_of_Device \" name= \"SenderPort \"/> \n \
90: \t \t<Variable format= \"UINT32 \" kind= \"ID \" name= \"SenderSensorID \"/> \n \
91: \t \t<Variable format= \"UINT32 \" kind= \"ID \" name= \"SDMTaskPID \" invalidValue= \"0 \"
description= \"The positive SDM process identifier number for a task. \"/> \n \
92: \t \t<Variable format= \"UINT16 \" kind= \"tbd \" name= \"NumTasks \"/> \n \
93: \t \t<Variable name= \"TaskListBlob \" format= \"UINT08 \" length= \"8067 \" kind= \"blob \"
description= \"A null separated list of UINT32/ASCII text pairs representing the PID and TaskName of
tasks in the requested list, respectively, ended with a double null-terminator. \"/> \n \
94: \t \t<Variable name= \"PmNodeId \" format= \"UINT08 \" kind= \"id \"/> \n \
95: \t \t<Variable name= \"killLevel \" format= \"UINT08 \" kind= \"id \" /> \n \
96: \t \t<Variable name= \"ExecutionMode \" format= \"INT08 \" kind= \"Mode \" description= \"Task
execution mode. \"> \n \
97: \t \t \t<Drange name= \"ExecutionModeEnum \"> \n \
98: \t \t \t \t<Option value= \"1 \" name= \"Normal \"/> \n \
99: \t \t \t \t<Option value= \"2 \" name= \"AlwaysRunning \"/> \n \
100: \t \t \t</Drange> \n \
101: \t \t</Variable> \n \
102: \t \t<Variable name= \"ArchitectureType \" format= \"INT08 \" kind= \"tbd \"> \n \
103: \t \t \t<Drange name= \"ArchitectureTypeEnum \"> \n \
104: \t \t \t \t<Option name= \"Intel \" value= \"1 \"/> \n \
105: \t \t \t \t<Option name= \"Microblaze \" value= \"2 \"/> \n \
106: \t \t \t</Drange> \n \
107: \t \t</Variable> \n \
108: \t \t<Variable name= \"OsType \" format= \"INT08 \" kind= \"tbd \"> \n \
109: \t \t \t<Drange name= \"OsTypeEnum \"> \n \
110: \t \t \t \t<Option name= \"Linux26 \" value= \"1 \"/> \n \
111: \t \t \t \t<Option name= \"Win32 \" value= \"2 \"/> \n \
112: \t \t \t</Drange> \n \

```

113: \t \t</Variable> \n \
 114: \t \t<Variable name= \"MemType \" format= \"INT08 \" kind= \"tbd \"> \n \
 115: \t \t \t<Drange name= \"MemTypeEnum \"> \n \
 116: \t \t \t \t<Option name= \"Mem64 \" value= \"1 \"/> \n \
 117: \t \t \t \t<Option name= \"Mem128 \" value= \"2 \"/> \n \
 118: \t \t \t \t<Option name= \"Mem256 \" value= \"3 \"/> \n \
 119: \t \t \t \t<Option name= \"Mem512 \" value= \"4 \"/> \n \
 120: \t \t \t</Drange> \n \
 121: \t \t</Variable> \n \
 122: \t \t<Variable name= \"TaskState \" format= \"INT08 \" kind= \"tbd \"> \n \
 123: \t \t \t<Drange name= \"TaskStateEnum \"> \n \
 124: \t \t \t \t<Option name= \"Queued \" value= \"1 \"/> \n \
 125: \t \t \t \t<Option name= \"Scheduled \" value= \"2 \"/> \n \
 126: \t \t \t \t<Option name= \"Running \" value= \"3 \"/> \n \
 127: \t \t \t \t<Option name= \"Stopped \" value= \"4 \"/> \n \
 128: \t \t \t</Drange> \n \
 129: \t \t</Variable> \n \
 130: \t \t<Variable name= \"TaskRetrievalLocation \" format= \"INT08 \" kind= \"tbd \"> \n \
 131: \t \t \t<Drange name= \"TaskLoadLocationEnum \"> \n \
 132: \t \t \t \t<Option value= \"1 \" name= \"PrimaryLocation \"/> \n \
 133: \t \t \t \t<Option value= \"2 \" name= \"TemporaryTestLocation \"/> \n \
 134: \t \t \t \t<Option value= \"3 \" name= \"BackupLocation \"/> \n \
 135: \t \t \t</Drange> \n \
 136: \t \t</Variable> \n \
 137: \n \t \t<Notification> \n \
 138: \t \t \t<DataMsg name= \"Status \" id= \"1 \" msgArrival= \"EVENT \"> \n \
 139: \t \t \t \t<VariableRef name= \"Mode \"/> \n \
 140: \t \t \t \t<VariableRef name= \"SenderAddr \"/> \n \
 141: \t \t \t \t<VariableRef name= \"SenderPort \"/> \n \
 142: \t \t \t \t<VariableRef name= \"SenderSensorID \"/> \n \
 143: \t \t \t</DataMsg> \n \
 144: \t \t</Notification> \n \
 145: \t \t<Notification> \n \
 146: \t \t \t<DataMsg name= \"TaskQueued \" id= \"2 \" msgArrival= \"EVENT \"> \n \
 147: \t \t \t \t<VariableRef name= \"TaskName \"/> \n \
 148: \t \t \t</DataMsg> \n \
 149: \t \t</Notification> \n \
 150: \t \t<Notification> \n \
 151: \t \t \t<DataMsg name= \"TaskStarted \" id= \"3 \" msgArrival= \"EVENT \"> \n \
 152: \t \t \t \t<VariableRef name= \"TaskName \"/> \n \
 153: \t \t \t \t<VariableRef name= \"SDMTaskPID \"/> \n \

154: \t \t \t</DataMsg> \n \\
 155: \t \t</Notification> \n \\
 156: \t \t<Notification> \n \\
 157: \t \t \t<DataMsg name= \"TaskFinished \" id= \"4\" msgArrival= \"EVENT \"> \n \\
 158: \t \t \t \t<VariableRef name= \"TaskName \"/> \n \\
 159: \t \t \t \t<VariableRef name= \"SDMTaskPID \"/> \n \\
 160: \t \t \t \t<VariableRef name= \"ExitStatus \"/> \n \\
 161: \t \t \t</DataMsg> \n \\
 162: \t \t</Notification> \n \\
 163: \t \t<Request> \n \\
 164: \t \t \t<CommandMsg name= \"GetTaskList \" id= \"10 \" description= \"Requests entire task list. \"/> \n \\
 165: \t \t \t<DataReplyMsg name= \"TaskListReply \" id= \"11 \"> \n \\
 166: \t \t \t \t<VariableRef name= \"NumTasks \"/> \n \\
 167: \t \t \t \t<VariableRef name= \"TaskListBlob \"/> \n \\
 168: \t \t \t</DataReplyMsg> \n \\
 169: \t \t</Request> \n \\
 170: \t \t<Request> \n \\
 171: \t \t \t<CommandMsg name= \"GetRunningTaskList \" id= \"15 \" description= \"Requests a list of running tasks. \"/> \n \\
 172: \t \t \t<DataReplyMsg name= \"RunningTaskListReply \" id= \"16 \"> \n \\
 173: \t \t \t \t<VariableRef name= \"NumTasks \"/> \n \\
 174: \t \t \t \t<VariableRef name= \"TaskListBlob \"/> \n \\
 175: \t \t \t</DataReplyMsg> \n \\
 176: \t \t</Request> \n \\
 177: \t \t<Request> \n \\
 178: \t \t \t<CommandMsg name= \"GetTaskInfo \" id= \"12 \"> \n \\
 179: \t \t \t \t<VariableRef name= \"SDMTaskPID \"/> \n \\
 180: \t \t \t</CommandMsg> \n \\
 181: \t \t \t<DataReplyMsg name= \"TaskInfoReply \" id= \"13 \"> \n \\
 182: \t \t \t \t<VariableRef name= \"SDMTaskPID \"/> \n \\
 183: \t \t \t \t<VariableRef name= \"TaskName \"/> \n \\
 184: \t \t \t \t<VariableRef name= \"ArchitectureType \"/> \n \\
 185: \t \t \t \t<VariableRef name= \"OsType \"/> \n \\
 186: \t \t \t \t<VariableRef name= \"MemType \"/> \n \\
 187: \t \t \t \t<VariableRef name= \"PmNodeId \"/> \n \\
 188: \t \t \t \t<VariableRef name= \"TaskState \"/> \n \\
 189: \t \t \t \t<VariableRef name= \"ExecutionMode \"/> \n \\
 190: \t \t \t</DataReplyMsg> \n \\
 191: \t \t</Request> \n \\
 192: \t \t<Request> \n \

```

193: \t \t \t<CommandMsg name= \"NameToPID \" id= \"6 \"> \n \
194: \t \t \t \t<VariableRef name= \"TaskName \"/> \n \
195: \t \t \t</CommandMsg> \n \
196: \t \t \t<DataReplyMsg name= \"TaskPID \" id= \"7 \"> \n \
197: \t \t \t \t<VariableRef name= \"TaskName \"/> \n \
198: \t \t \t \t<VariableRef name= \"SDMTaskPID \"/> \n \
199: \t \t \t</DataReplyMsg> \n \
200: \t \t</Request> \n \
201: \t \t<Command> \n \
202: \t \t \t<CommandMsg name= \"ChangeMode \" id= \"5 \"> \n \
203: \t \t \t \t<VariableRef name= \"Mode \"/> \n \
204: \t \t \t \t<VariableRef name= \"SenderAddr \"/> \n \
205: \t \t \t \t<VariableRef name= \"SenderPort \"/> \n \
206: \t \t \t \t<VariableRef name= \"SenderSensorID \"/> \n \
207: \t \t \t</CommandMsg> \n \
208: \t \t</Command> \n \
209: \t \t<Command> \n \
210: \t \t \t<CommandMsg name= \"StartTask \" id= \"8 \"> \n \
211: \t \t \t \t<VariableRef name= \"TaskName \"/> \n \
212: \t \t \t \t<VariableRef name= \"ArchitectureType \"/> \n \
213: \t \t \t \t<VariableRef name= \"OsType \"/> \n \
214: \t \t \t \t<VariableRef name= \"MemType \"/> \n \
215: \t \t \t \t<VariableRef name= \"ExecutionMode \"/> \n \
216: \t \t \t \t<VariableRef name= \"PmNodeId \"/> \n \
217: \t \t \t \t<VariableRef name= \"TaskRetrievalLocation \"/> \n \
218: \t \t \t</CommandMsg> \n \
219: \t \t</Command> \n \
220: \t \t<Command> \n \
221: \t \t \t<CommandMsg name= \"KillTask \" id= \"9 \"> \n \
222: \t \t \t \t<VariableRef name= \"SDMTaskPID \"/> \n \
223: \t \t \t \t<VariableRef name= \"killLevel \"/> \n \
224: \t \t \t</CommandMsg> \n \
225: \t \t</Command> \n \
226: \t</Interface> \n \
227: \t<Interface name= \"Msg_Count \" id= \"2 \"> \n \
228: \t \t<Variable name= \"Total_Messages_Recd \" kind= \"Total \" format= \"UINT32 \"/> \n \
229: \t \t<Variable name= \"Messages_Last_Second_Recd \" kind= \"Total \" format= \"UINT32 \"/> \n \
230: \t \t<Variable name= \"Total_Messages_Sent \" kind= \"Total \" format= \"UINT32 \"/> \n \
231: \t \t<Variable name= \"Messages_Last_Second_Sent \" kind= \"Total \" format= \"UINT32 \"/> \n \
232: \t \t<Notification> \n \

```

```

233: \t \t \t<DataMsg name= \"Message_Count \" id= \"13 \" msgArrival= \"PERIODIC \"> \n \
234: \t \t \t \t<VariableRef name= \"Total_Messages_Recd \"/> \n \
235: \t \t \t \t<VariableRef name= \"Messages_Last_Second_Recd \"/> \n \
236: \t \t \t \t<VariableRef name= \"Total_Messages_Sent \"/> \n \
237: \t \t \t \t<VariableRef name= \"Messages_Last_Second_Sent \"/> \n \
238: \t \t \t</DataMsg> \n \
239: \t \t</Notification> \n \
240: \t</Interface> \n \
241: \t<Interface name= \"Message_Log \" id= \"3 \"> \n \
242: \t \t<Variable format= \"UINT08 \" kind= \"TBD \" name= \"Msg_Type \"/> \n \
243: \t \t<Variable format= \"UINT32 \" kind= \"IP_long \" name= \"Address \"/> \n \
244: \t \t<Variable format= \"UINT16 \" kind= \"Port_of_Device \" name= \"Port \"/> \n \
245: \t \t<Variable format= \"UINT32 \" kind= \"ID \" name= \"Sensor_ID \"/> \n \
246: \t \t<Command> \n \
247: \t \t \t<CommandMsg name= \"Enable_Logging \" id= \"16 \"> \n \
248: \t \t \t \t<VariableRef name= \"Msg_Type \"/> \n \
249: \t \t \t \t<VariableRef name= \"Address \"/> \n \
250: \t \t \t \t<VariableRef name= \"Port \"/> \n \
251: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
252: \t \t \t</CommandMsg> \n \
253: \t \t</Command> \n \
254: \t \t<Command> \n \
255: \t \t \t<CommandMsg name= \"Disable_Logging \" id= \"17 \"> \n \
256: \t \t \t \t<VariableRef name= \"Msg_Type \"/> \n \
257: \t \t \t \t<VariableRef name= \"Address \"/> \n \
258: \t \t \t \t<VariableRef name= \"Port \"/> \n \
259: \t \t \t \t<VariableRef name= \"Sensor_ID \"/> \n \
260: \t \t \t</CommandMsg> \n \
261: \t \t</Command> \n \
262: \t</Interface> \n \
263: </xTEDS>\"
264:
265: #endif // #ifndef SDM_TM_XTEDS_DEFS_H_

```

File: sdm/tm/pm_record.h

```
1: #ifndef SDM_PM_RECORD
2: #define SDM_PM_RECORD
3:
4: #include "../common/message/SDMComponent_ID.h"
5: #include "../common/task/SDMTaskResources.h"
6:
7: class pm_record
8: {
9: public:
10: pm_record();
11: pm_record(const pm_record &b);
12: pm_record& operator=(const pm_record& b);
13: ~pm_record();
14: SDMComponent_ID component_id;
15: SDMTaskResources resources;
16: short tasks;
17: bool in_use;
18: };
19:
20: #endif
```


File: sdm/tm/pm_record_list.h

```
1: #ifndef _SDM_PM_RECORD_LIST_H_
2: #define _SDM_PM_RECORD_LIST_H_
3:
4: #include <pthread.h>
5: #include "pm_record.h"
6: #include "../common/message/SDMComponent_ID.h"
7: #include "../common/message/SDMHeartbeat.h"
8: #include "../common/task/SDMTaskResources.h"
9:
10: #define HEARTBEAT_INACTIVE -1
11: #define MAX_PM 8
12:
13: // This class used to hold records for Process Managers used with the Task Manager. All
14: // PMs are generally identified with the TM by the PM component identifier. This
15: // class is thread-safe.
16:
17: class PMRecordList
18: {
19: public:
20: PMRecordList();
21: // List related functions
22: bool RegisterPM(const SDMComponent_ID& PMAAddress, const SDMTaskResources& Resources);
23: bool AlreadyRegistered(const SDMComponent_ID& PMAAddress, const SDMTaskResources&
Resources = 0);
24: void ClearAllRunningTasks();
25: bool TaskFinished(const SDMComponent_ID& PMAAddress);
26: bool TaskHasStarted(const SDMComponent_ID& PMAAddress);
27: bool RemovePM();
28: bool FindPMID(const SDMTaskResources& TaskResources, pm_record& PmDataOut);
29: bool FindEligiblePM(int NumTasks, const SDMTaskResources& TaskResources, pm_record&
PmDataOut);
30: void SetTasks(const SDMComponent_ID& PMAAddress, int NumTasks);
31: int GetTasks(const SDMComponent_ID& PMAAddress);
32: void SendToAll(SDMmessage& message);
33:
34: bool IsEmpty();
35:
36: // Heartbeat related functions
37: int SendHeartbeatToAll(SDMHeartbeat& HearbeatMessage);
```

```
38: void HeartbeatReceived(const SDMComponent_ID& PMAddress);
39: bool RemoveAnyFailed(pm_record &PMOut);
40:
41: void PrintList();
42: private:
43: pm_record pmList[MAX_PM];
44: unsigned int schedulerInfoIndex;
45: int HeartbeatStatus[MAX_PM];
46: pthread_mutex_t DataMutex;
47: };
48:
49:
50: #endif
```

File: sdm/tm/tm.h

```
1: #ifndef _SDM_TM_H_
2: #define _SDM_TM_H_
3:
4: #include <pthread.h>
5: #include <stdio.h>
6: #include <stdlib.h>
7: #include <unistd.h>
8: #include <fcntl.h>
9: #include <errno.h>
10: #include <arpa/inet.h>
11: #ifndef __VXWORKS__
12: #include <getopt.h>
13: #include <sys/poll.h>
14: #endif
15: #include <signal.h>
16: #include <sys/time.h>
17: #include <sys/types.h>
18: #include <sys/stat.h>
19: #include <sys/ioctl.h>
20:
21:
22: #ifndef WIN32
23: #include <net/if.h>
24: #endif
25:
26: #include "tasklist.h"
27: #include "../common/UDPcom.h"
28: #include "../common/checksum/checksum.h"
29: #include "../common/message/SDMError.h"
30: #include "../common/message/SDMPostTask.h"
31: #include "../common/message/SDMxTEDS.h"
32: #include "../common/message/SDMSubreqst.h"
33: #include "../common/message/SDMDeletesub.h"
34: #include "../common/message/SDMCommand.h"
35: #include "../common/message/SDMReady.h"
36: #include "../common/message/SDMTask.h"
37: #include "../common/message/SDMCancelxTEDS.h"
38: #include "../common/message/SDMRegPM.h"
39: #include "../common/message/SDMCode.h"
```

```

40: #include "../common/message/SDMReqCode.h"
41: #include "../common/message/SDMTaskFinished.h"
42: #include "../common/message/SDMHeartbeat.h"
43: #include "../common/message/SDMKill.h"
44: #include "../common/message/SDMRegInfo.h"
45: #include "../common/message/SDMReqReg.h"
46: #include "../common/message/SDMDMLLeader.h"
47: #include "../common/message/SDMHello.h"
48: #include "../common/message/SDMAck.h"
49: #include "../common/message/SDMRegister.h"
50: #include "../common/message/SDMID.h"
51:
52: #include "../common/SubscriptionManager/SubscriptionManager.h"
53: #include "../common/MessageManager/MessageManager.h"
54: #include "../common/MessageLogger/MessageLogger.h"
55: #include "../common/MessageManipulator/MessageManipulator.h"
56: #include "../common/Debug.h"
57: #include "../common/version.h"
58: #include "../common/MemoryUtils.h"
59: #include "../common/Time/SDMTime.h"
60: #include "../dm/DMUtils.h"
61: #include "pm_record.h"
62: #include "pm_record_list.h"
63: #include "HandlerArguments.h"
64: #include "TmXtedsDefs.h"
65:
66: #ifdef __uClinux__
67: # define BUILD_FOR_PNPSAT
68: #endif
69:
70: #define MAX_JOBS 50
71:
72: //
73: // Definitions for task load locations
74: //
75: #ifdef __uClinux__
76: # define STR_TASK_LOCATION_PRIMARY "/mnt/flash/bin"
77: # define STR_TASK_LOCATION_TEMPORARY  "/mnt/flash/bin_test"
78: # define STR_TASK_LOCATION_BACKUP  "/mnt/flash/bin_backup"
79: #else
80: # define STR_TASK_LOCATION_PRIMARY "."

```

```

81: # define STR_TASK_LOCATION_TEMPORARY  "."
82: # define STR_TASK_LOCATION_BACKUP  "."
83: #endif // ifdef __uLinux__
84:
85: // Function prototypes
86: void printResources(const SDMTaskResources& resources);
87: bool SendTask (const pm_record& PMData, const Task& TaskData);
88: void SendCode (char* buf);
89: void PostTask (char *buf);
90: void VerifyDM(void);
91: void PublishStatusMessage(char *sender_buf);
92: void PublishTaskQueuedMessage(char* taskname);
93: void PublishTaskStartedMessage(char* taskname, unsigned int pid);
94: void PublishTaskFinishedMessage(char* taskname,int exitcode, unsigned int pid);
95: void PublishPerformanceCounterMessage(void);
96: void InitialStartUp(void);
97: void MessageReceived(SDMmessage *msg);
98: void MessageSent(SDMmessage *msg);
99: void* Listener(void*);
100: void* Scheduler(void*);
101: void* TMHeartbeat(void*);
102: void* PMHeartbeat(void*);
103: void* ReqCodeHandler(void* arg);
104: void* PostTaskHandler(void* arg);
105: void* SubreqstHandler(void* arg);
106: void* DeletesubHandler(void* arg);
107: void* CommandHandler(void* arg);
108: void* SerreqstHandler(void* arg);
109: void* ReadyHandler(void* arg);
110: void RegInfoHandler(void* arg);
111: void* TaskFinishedHandler(void* arg);
112: void* KillHandler(void* arg);
113: bool PerformKill(SDMKill& msgKill);
114: void* PMRunningTaskHandler(void* arg);
115: void QueueHeartbeat (long length, char buf[]);
116: void ReceiveLeader (char buf[]);
117: void TakeOverLeadPosition (void);
118: double GetCurTime();
119: #ifndef WIN32
120: void* SigHandler(void*);
121: #else

```

```
122: void SigHandler(int signum);
123: #endif
124:
125: #ifdef PNP_FAKE
126: void sendIMA (void);
127: void RequestDMInfo(MessageManager*);
128: void AmIBackup(MessageManager*);
129: #endif
130:
131: #ifdef PNP_BACKUP
132: void RequestDMInfo(void);
133: bool AmIBackup (void);
134: void VerifyNewDM(char *sender_buf);
135: #endif
136:
137: #ifdef BUILD_FOR_PNPSAT
138: void RequestRegPowerController();
139: void PowerCyclePmNode(const SDMComponent_ID& idPm);
140: #endif // BUILD_FOR_PNPSAT
141:
142: #endif // _SDM_TM_H_
```

File: sdm/tm/tm_monitor_win32.cpp

```
1: //*****
2: //tm_monitor
3: //
4: //The tm_monitor adds robustness to the TaskManager. The monitor periodically sends
5: //heartbeat messages to the TaskManager module residing on the same node. If for
6: //some reason, the TM fails to respond, the monitor attempts to restart it.
7: //*****
8:
9: #include <stdio.h>
10: #include <unistd.h>
11: #include <sys/types.h>
12: #include <signal.h>
13: #include <stdlib.h>
14: #include <arpa/inet.h>
15: #include <netinet/in.h>
16: #include <sys/socket.h>
17: #include <sys/wait.h>
18: #include <string.h>
19: #include "../common/message/SDMReady.h"
20: #include "../common/MessageManager/MessageManager.h"
21: #include "../common/TCPcom.h"
22: #include <windows.h>
23: #include <process.h>
24:
25: #define NUM_HEARTBEAT_TRIES 2 /*Number of times to miss a response before restarting
the TM*/
26:
27: HANDLE StartTM(char **);
28: void SigIntHandler(int);
29:
30: HANDLE tm_handle = NULL; /* Process handle of the TaskManager */
31:
32: int main(int argc, char ** argv)
33: {
34: unsigned long result = 0;
35: int length = 0;
36: int miss_count = 0;
37: int num_toget = 0;
38: char buf[BUFSIZE];
```

```

39: SDMReady heartbeat;
40: MessageManager mm;
41: SDMComponent_ID tm;
42:
43: //Set monitor process address
44: heartbeat.source.setSensorID(0);
45: heartbeat.source.setAddress(inet_addr("127.0.0.1"));
46: heartbeat.source.setPort(PORT_TM_MONITOR);
47:
48: //Set TM address
49: tm.setAddress(inet_addr("127.0.0.1"));
50: tm.setPort(PORT_TM);
51: tm.setSensorID(0);
52:
53: signal(SIGINT, SigIntHandler);
54: tm_handle = StartTM(argv);
55: mm.Async_Init_Both(PORT_TM_MONITOR);
56:
57: /* If spawn was successful */
58: if (tm_handle != NULL)
59: {
60:     /* Allow the TM to get started up */
61:     sleep(HEARTBEAT_INTERVAL);
62:     while (1)
63:     {
64:         //Send heartbeats via UDP
65:         heartbeat.SendTo(tm);
66:         length = heartbeat.Marshal(buf);
67:         sleep(HEARTBEAT_INTERVAL);
68:
69:         //If Task Manager quit
70:         if (!GetExitCodeProcess(tm_handle,&result))
71:             printf(" Monitor: Invalid handle for retrieving exit code. \n");
72:         /* If the TaskManager has quit for some reason */
73:         if (result != STILL_ACTIVE)
74:         {
75:             printf(" Monitor: TM failed, restarting... \n");
76:             tm_handle = StartTM(argv);
77:             if (tm_handle != NULL)
78:             {
79:                 /* Allow the TM to restart */

```



```

80:         sleep(HEARTBEAT_INTERVAL);
81:         /* Start over at while */
82:         continue;
83:     }
84:     /* Error re-starting TM */
85:     else
86:     {
87:         printf(" Monitor: Could not restart the TM. \n");
88:         return -1;
89:     }
90: }
91: /* The TM is still running, check for responses to heartbeat messages */
92: else if (mm.IsReady())
93: {
94:     //Should respond with one message
95:     num_toget = 1;
96:
97:     while (mm.IsReady())
98:     {
99:         num_toget--;
100:        /* Message is discarded, we only care that one was received */
101:        mm.GetMsg(buf);
102:    }
103:    if (num_toget > 0)
104:        miss_count++;
105:    else
106:        miss_count = 0;
107: }
108: /* The TM is still running, but no responses were received */
109: else
110: {
111:     //If no messages were received
112:     miss_count++;
113:     if (miss_count == NUM_HEARTBEAT_TRIES)
114:     {
115:         printf(" Monitor: TM unresponsive, restarting... \n");
116:         /* If the Terminate fails and the TM is already dead */
117:         if (!TerminateProcess(tm_handle,0))
118:         {
119:             /* Get its return value */
120:             GetExitCodeProcess(tm_handle,&result);

```

```

121:         }
122:         /* Restart the TM */
123:         tm_handle = StartTM(argv);
124:         /* Success restarting */
125:         if (tm_handle != NULL)
126:         {
127:             /* Allow TM to get started */
128:             sleep(HEARTBEAT_INTERVAL);
129:             /* Start over at while loop */
130:             continue;
131:         }
132:         /* Error restarting */
133:         else
134:         {
135:             printf(" Monitor: Could not restart the TM. \n");
136:             return -1;
137:         }
138:     }
139: }
140: }
141: }
142: else
143: {
144:     printf(" Monitor: Error starting the TM. \n");
145:     return -1;
146: }
147: return 0;
148: }
149:
150: /*

```

151: StartTM starts the TaskManager module of the SDM from the monitor process. All command line arguments are passed from the

152: monitor process. This function assumes the that both the TaskManager and the monitor binaries reside in the same directory.

153: INPUTS:

154: argv - The command line arguments passed to the TaskManager.

155: RETURNS:

156: HANDLE - The handle to the spawned TaskManager, or NULL if an error has occurred.

157: */

158: HANDLE StartTM(char ** argv)

159: {

```

160: int p_handle;                                /* Handle to the spawned process (TaskManager) */
161: char proc_name[512];                          /* Buffer for command to execute */
162: memset(proc_name,'\\0',sizeof(proc_name)); /* Clear the char buffer */
163:
164: /* Assure that arguments have been passed */
165: if (argv)
166: {
167:     /*argv[0] contains the binary name from the command line */
168:     /*If invoked on command line without .exe */
169:     if (strstr(argv[0],".exe") == NULL)
170:         strncpy(proc_name,argv[0],strlen(argv[0])-8); /* Copying everything but _monitor */
171:     /*If invoked with .exe */
172:     else
173:         strncpy(proc_name,argv[0],strlen(argv[0])-12); /* Copying everything but _monitor.exe */
174:     strcat(proc_name, ".exe"); /* String is now ...tm.exe */
175: }
176: /* If no command line arguments, error */
177: else
178:     return NULL;
179: /* Start the TaskManager */
180: p_handle = _spawnv(_P_NOWAIT,proc_name,argv);
181: /* If error */
182: if (p_handle == -1)
183: {
184:     perror("");
185:     return NULL;
186: }
187: /* Return the handle */
188: else
189:     return (HANDLE)p_handle;
190: }
191: /*
192:  SigIntHandler handles a CTRL+C (SIGINT) signal. This function will kill the TaskManager and
the monitor process.
193:  INPUTS:
194:      sig_num - The signal sent.
195:  RETURNS:
196:      void - None.
197: */
198: void SigIntHandler(int sig_num)

```

```
199: {
200:   unsigned long result = 0;
201:   if (!tm_handle)
202:     exit(EXIT_SUCCESS);
203:   sleep(1);
204:   if (!TerminateProcess(tm_handle,0))
205:     GetExitCodeProcess(tm_handle,&result);
206:   exit(EXIT_SUCCESS);
207: }
208:
```

File: sdm/tm/tm_monitor.cpp

```
1: #include <stdio.h>
2: #include <unistd.h>
3: #include <sys/types.h>
4: #include <signal.h>
5: #include <stdlib.h>
6: #include <arpa/inet.h>
7: #include <netinet/in.h>
8: #include <sys/socket.h>
9: #include <sys/wait.h>
10: #include <string.h>
11: #include "../common/message/SDMReady.h"
12: #include "../common/MessageManager/MessageManager.h"
13: #include "../common/TCPcom.h"
14:
15: #define NUM_HEARTBEAT_TRIES 10 /*Number of times to miss a response before restarting
the TM*/
16:
17: int StartTM(char **);
18: void SigIntHandler(int);
19:
20: int tm_pid = -1;    //Process id number of the Task Manager
21:
22: int main(int argc, char ** argv)
23: {
24:     int result = -1;
25:     int length = 0;
26:     int miss_count = 0;
27:     int num_toget = 0;
28:     char buf[BUFSIZE];
29:     SDMReady heartbeat;
30:     MessageManager mm;
31:     SDMComponent_ID tm;
32:
33:     if (argc == 1)
34:     {
35:         printf("Usage: \n%s tm_process [tm_process_args...] \n", argv[0]);
36:         return -1;
37:     }
38:
```

```

39: // Set up the command line for the child process
40: char** nargv = new char*[argc];
41: for (int i = 1; i < argc; ++i)
42:     nargv[i-1] = argv[i];
43: nargv[argc-1] = NULL;
44:
45: // Set up TM arguments for a restart after a fault
46: char fault_string[3] = "-f";
47: char **fault_args = new char*[argc+1];
48: for (int i = 1; i < argc; ++i)
49:     fault_args[i-1] = argv[i];
50: fault_args[argc-1] = fault_string;
51: fault_args[argc] = NULL;
52:
53: //Set monitor process address
54: heartbeat.source.setSensorID(0);
55: heartbeat.source.setAddress(inet_addr("127.0.0.1"));
56: heartbeat.source.setPort(PORT_TM_MONITOR);
57:
58: //Set TM address
59: tm.setAddress(inet_addr("127.0.0.1"));
60: tm.setPort(PORT_TM);
61: tm.setSensorID(0);
62:
63: signal(SIGINT, SigIntHandler);
64: tm_pid = StartTM(nargv);
65: mm.Async_Init(PORT_TM_MONITOR);
66:
67: //If fork/exec was successful
68: if (tm_pid > 0)
69: {
70:     //Allow the TM to get started up
71:     sleep(HEARTBEAT_INTERVAL);
72:     while (1)
73:     {
74:
75:         //Send heartbeats via UDP
76:         heartbeat.SendTo(tm);
77:         length = heartbeat.Marshal(buf);
78:         sleep(HEARTBEAT_INTERVAL);
79:

```

```

80:     //If Task Manager quit
81:     if (waitpid(-1,&result,WNOHANG) == tm_pid)
82:     {
83:         printf(" Monitor: TM failed, restarting... \n");
84:         tm_pid = StartTM(fault_args);
85:         if (tm_pid > 0)
86:         {
87:             sleep(HEARTBEAT_INTERVAL);
88:             continue;
89:         }
90:         else
91:         {
92:             printf(" Monitor: Could not restart the TM. \n");
93:             return -1;
94:         }
95:     }
96:     else if (mm.IsReady())
97:     {
98:         //Should respond with two messages
99:         num_toget = 1;
100:
101:         while (mm.IsReady())
102:         {
103:             num_toget--;
104:             mm.GetMessage(buf);
105:         }
106:         if (num_toget > 0)
107:             miss_count++;
108:         else
109:             miss_count = 0;
110:     }
111:     else
112:     {
113:         //If no messages were received
114:         miss_count++;
115:         if (miss_count == NUM_HEARTBEAT_TRIES)
116:         {
117:             printf(" Monitor: TM unresponsive, restarting... \n");
118:
119:             if (kill (tm_pid, SIGKILL) == 0)
120:                 wait(&result);

```

```

121:         else
122:             waitpid(-1, &result, WNOHANG);
123:
124:             tm_pid = StartTM(fault_args);
125:             if (tm_pid > 0)
126:             {
127:                 sleep(HEARTBEAT_INTERVAL);
128:                 continue;
129:             }
130:             else
131:             {
132:                 printf(" Monitor: Could not restart the TM. \n");
133:                 return -1;
134:             }
135:
136:
137:         }
138:     }
139: }
140: }
141: else
142: {
143:     printf(" Monitor: Error starting the TM (%d). \n", tm_pid);
144:     return -1;
145: }
146:
147:
148: return 0;
149: }
150:
151: int StartTM(char ** argv)
152: {
153:     int pid;
154:
155:     pid = vfork();
156:     //Child Process
157:     if (pid == 0)
158:     {
159:         //Start the Task Manager
160:         if (execvp(argv[0],argv) < 0)
161:         {

```



```

162:         printf(" Monitor: Error exec'ing the TM. \n");
163:         exit(-1);
164:     }
165:     //Here to make the compiler happy, never runs however
166:     return 0;
167: }
168: //Parent Process
169: else
170:     return pid;
171: }
172:
173: void SigIntHandler(int sig_num)
174: {
175:     int result = 0;
176:     if (kill (SIGINT, tm_pid) == 0)
177:         wait (&result);
178:     else
179:         waitpid(-1,&result,WNOHANG);
180:
181:     exit(EXIT_SUCCESS);
182: }
183:

```

File: sdm/tm/pm_record_list.cpp

```
1: #include "pm_record_list.h"
2: #include "task.h"
3: #include <stdio.h>
4:
5: PMRecordList::PMRecordList(): schedulerInfoIndex(0), DataMutex()
6: {
7:     pthread_mutex_init(&DataMutex, NULL);
8:     for (unsigned int i = 0; i < MAX_PM; i++)
9:         HeartbeatStatus[i] = 0;
10: }
11:
12: // Determine whether a PM with the same address is already in the list
13: bool PMRecordList::AlreadyRegistered(const SDMComponent_ID& PMAddress, const
SDMTaskResources& Resources)
14: {
15:     pthread_mutex_lock(&DataMutex);
16:     for (unsigned int i = 0; i < MAX_PM; i++)
17:     {
18:         if (pmList[i].in_use)
19:         {
20:             if (pmList[i].component_id.getAddress() == PMAddress.getAddress())
21:             {
22:                 pthread_mutex_unlock(&DataMutex);
23:                 return true;
24:             }
25:             if (Resources.IsPreferredPmIdSet())
26:             {
27:                 if (pmList[i].resources.GetPreferredPmId() == Resources.GetPreferredPmId())
28:                     printf("Warning: Multiple PM's registered with same PM node ID. \n");
29:             }
30:         }
31:     }
32:     pthread_mutex_unlock(&DataMutex);
33:     return false;
34: }
35:
36: bool PMRecordList::RegisterPM(const SDMComponent_ID& PMAddress, const
SDMTaskResources& Resources)
37: {
```

```

38: // Find an unused PM entry
39: unsigned int i;
40:
41: pthread_mutex_lock(&DataMutex);
42: for (i = 0; i < MAX_PM; i++)
43: {
44:     if (!pmList[i].in_use)
45:         break;
46: }
47: if (i == MAX_PM)
48: {
49:     pthread_mutex_unlock(&DataMutex);
50:     return false; // No more entries available
51: }
52:
53: pmList[i].in_use = true;
54: pmList[i].tasks = 0;
55: pmList[i].resources = Resources;
56: pmList[i].component_id = PMAddress;
57: // Pms always have PORT_PM
58: pmList[i].component_id.setPort(PORT_PM);
59:
60: pthread_mutex_unlock(&DataMutex);
61: return true;
62: }
63:
64: // Clears all running tasks for each PM, useful for when TM resets
65: void PMRecordList::ClearAllRunningTasks()
66: {
67:     pthread_mutex_lock(&DataMutex);
68:     for (unsigned int i = 0; i < MAX_PM; i++)
69:     {
70:         // No need to check for in_use, setting all to zero will have no negative effect
71:         pmList[i].tasks = 0;
72:     }
73:     pthread_mutex_unlock(&DataMutex);
74: }
75:
76: // Decrement the number of running tasks for the given PM
77: bool PMRecordList::TaskFinished(const SDMComponent_ID& PMAddress)
78: {

```

```

79: pthread_mutex_lock(&DataMutex);
80: for (unsigned int i = 0; i < MAX_PM; i++)
81: {
82:     if (pmList[i].in_use && pmList[i].component_id.getAddress() == PMAccess.getAddress())
83:     {
84:         pmList[i].tasks--;
85:         pthread_mutex_unlock(&DataMutex);
86:         return true;
87:     }
88: }
89: pthread_mutex_unlock(&DataMutex);
90: return false;
91: }
92:
93: bool PMRecordList::TaskHasStarted(const SDMComponent_ID& PMAccess)
94: {
95: pthread_mutex_lock(&DataMutex);
96: for (unsigned int i = 0; i < MAX_PM; i++)
97: {
98:     if (pmList[i].in_use && pmList[i].component_id.getAddress() == PMAccess.getAddress())
99:     {
100:         pmList[i].tasks++;
101:         pthread_mutex_unlock(&DataMutex);
102:         return true;
103:     }
104: }
105: pthread_mutex_unlock(&DataMutex);
106: return false;
107: }
108:
109: bool PMRecordList::FindPMID(const SDMTaskResources& TaskResources, pm_record&
PmDataOut)
110: {
111:     if (!TaskResources.IsPreferredPmIdSet())
112:         return false;
113:
114:     // Otherwise, find the PM with the matching id
115:     pthread_mutex_lock(&DataMutex);
116:     for (unsigned int i = 0; i < MAX_PM; i++)
117:     {
118:         if (TaskResources.GetPreferredPmId() == pmList[i].resources.GetPreferredPmId())

```

```

119:     {
120:         PmDataOut = pmList[i];
121:         pthread_mutex_unlock(&DataMutex);
122:         return true;
123:     }
124: }
125: pthread_mutex_unlock(&DataMutex);
126:
127: // The preferred PM has not registered
128: return false;
129: }
130:
131: // Find an eligible PM for scheduling and return it in PmDataOut. A PM is only eligible if its
132: // number of running tasks is no greater than NumTasks. Keep the scheduler index as a member
133: // variable to prevent looking at the same PM if the resources don't allow a schedule match.
134: bool PMRecordList::FindEligiblePM(int NumTasks, const SDMTaskResources& TaskResources,
pm_record& PmDataOut)
135: {
136:     ushort taskResources = TaskResources.GetIgnorePmId();
137:     ushort taskArchOs = taskResources&(ARCHMASK|OSMASK);
138:     pthread_mutex_lock(&DataMutex);
139:     for (; schedulerInfoIndex < MAX_PM; schedulerInfoIndex++)
140:     {
141:         ushort pmResources = pmList[schedulerInfoIndex].resources.GetIgnorePmId();
142:         if (pmList[schedulerInfoIndex].in_use && pmList[schedulerInfoIndex].tasks <= NumTasks
&&
143:             (pmResources&(ARCHMASK|OSMASK)) == taskArchOs &&
(pmResources&MEMMASK) >= (taskResources&MEMMASK) )
144:         {
145:             PmDataOut = pmList[schedulerInfoIndex];
146:             schedulerInfoIndex++;
147:             pthread_mutex_unlock(&DataMutex);
148:             return true;
149:         }
150:     }
151:     if (schedulerInfoIndex >= MAX_PM)
152:         schedulerInfoIndex = 0;
153:
154:     pthread_mutex_unlock(&DataMutex);
155:     return false;
156: }
157:

```

```

158: int PMRecordList::SendHeartbeatToAll(SDMHeartbeat& HeartbeatMessage)
159: {
160:     int numSent = 0;
161:
162:     pthread_mutex_lock(&DataMutex);
163:     for (unsigned int i = 0; i < MAX_PM; i++)
164:     {
165:         if (pmList[i].in_use)
166:         {
167:             HeartbeatMessage.SendTo(pmList[i].component_id);
168:             HeartbeatStatus[i]++;
169:             numSent++;
170:         }
171:         else
172:         {
173:             HeartbeatStatus[i] = HEARTBEAT_INACTIVE;
174:         }
175:     }
176:     pthread_mutex_unlock(&DataMutex);
177:     return numSent;
178: }
179:
180: void PMRecordList::SendToAll(SDMmessage& message)
181: {
182:     pthread_mutex_lock(&DataMutex);
183:
184:     for (unsigned int i = 0; i < MAX_PM; i++)
185:     {
186:         if (pmList[i].in_use)
187:         {
188:             message.SendTo(pmList[i].component_id);
189:         }
190:     }
191:
192:     pthread_mutex_unlock(&DataMutex);
193: }
194:
195: void PMRecordList::HeartbeatReceived(const SDMComponent_ID& PMAAddress)
196: {
197:     pthread_mutex_lock(&DataMutex);
198:     for (unsigned int i = 0; i < MAX_PM; i++)

```

```

199:  {
200:      if (pmList[i].in_use && pmList[i].component_id.getAddress() == PMAAddress.getAddress())
201:          HeartbeatStatus[i] = 0;
202:  }
203:  pthread_mutex_unlock(&DataMutex);
204: }
205:
206: bool PMRecordList::RemoveAnyFailed(pm_record &PMNodeOut)
207: {
208:     pthread_mutex_lock(&DataMutex);
209:     for (unsigned int i = 0; i < MAX_PM; i++)
210:     {
211:         if (pmList[i].in_use && HeartbeatStatus[i] >= 2)
212:         {
213:             // This node has failed
214:             PMNodeOut = pmList[i];
215:             pmList[i].in_use = false;
216:             pmList[i].tasks = 0;
217:             pmList[i].resources = 0;
218:             pthread_mutex_unlock(&DataMutex);
219:             return true;
220:         }
221:     }
222:     pthread_mutex_unlock(&DataMutex);
223:     return false;
224: }
225:
226: int PMRecordList::GetTasks(const SDMComponent_ID& PMAAddress)
227: {
228:     pthread_mutex_lock(&DataMutex);
229:     for (unsigned int i = 0; i < MAX_PM; i++)
230:     {
231:         if (pmList[i].in_use && pmList[i].component_id.getAddress() == PMAAddress.getAddress())
232:         {
233:             int iCurTasks = pmList[i].tasks;
234:             pthread_mutex_unlock(&DataMutex);
235:             return iCurTasks;
236:         }
237:     }
238:     pthread_mutex_unlock(&DataMutex);
239:     return 0;

```

```

240: }
241:
242: bool PMRecordList::IsEmpty()
243: {
244:     bool bAnyFound = false;
245:     pthread_mutex_lock(&DataMutex);
246:     for (unsigned int i = 0; i < MAX_PM; i++)
247:     {
248:         if (pmList[i].in_use)
249:         {
250:             bAnyFound = true;
251:             break;
252:         }
253:     }
254:     pthread_mutex_unlock(&DataMutex);
255:
256:     return !bAnyFound;
257: }
258:
259: void PMRecordList::SetTasks(const SDMComponent_ID& PMAAddress, int NumTasks)
260: {
261:     pthread_mutex_lock(&DataMutex);
262:     for (unsigned int i = 0; i < MAX_PM; i++)
263:     {
264:         if (pmList[i].in_use && pmList[i].component_id.getAddress() == PMAAddress.getAddress())
265:             pmList[i].tasks = NumTasks;
266:     }
267:     pthread_mutex_unlock(&DataMutex);
268: }
269:
270: void PMRecordList::PrintList()
271: {
272:     printf("***PMRecordList is \n");
273:
274:     pthread_mutex_lock(&DataMutex);
275:     for (unsigned int i = 0; i < MAX_PM; i++)
276:     {
277:         if (pmList[i].in_use)
278:         {
279:             char ID[128];
280:             pmList[i].component_id.IDToString(ID, sizeof(ID));

```



```
281:         printf("          Index    %u:    %s    tasks:    %hd    resources:    %hu\n",i,ID,pmList[i].tasks,pmList[i].resources.GetUShort());
282:     }
283: }
284: pthread_mutex_unlock(&DataMutex);
285: printf("**** \n \n");
286: }
287:
```

File: sdm/tm/Makefile.uclinux

```
1: ifndef PETALINUX
2: $(error You must source the petalinux/settings.sh script before working with PetaLinux)
3: endif
4:
5: # Point to default PetaLinux root directory
6: ifndef ROOTDIR
7: ROOTDIR=$(PETALINUX)/software/petalinux-dist
8: endif
9:
10: PATH:=$(PATH):$(ROOTDIR)/tools
11:
12: UCLINUX_BUILD_USER = 1
13: -include $(ROOTDIR)/.config
14: -include $(ROOTDIR)/$(CONFIG_LINUXDIR)/.config
15: LIBCDIR = $(CONFIG_LIBCDIR)
16: -include $(ROOTDIR)/config.arch
17: ROMFSDIR=$(ROOTDIR)/romfs
18: ROMFSINST=$(ROOTDIR)/tools/romfs-inst.sh
19:
20: APP = tm
21: MONITOR_APP = tm_monitor
22:
23: # Add any other object files to this list below
24: APP_OBJS = tm.o pm_record.o pm_record_list.o TMUtil.o task.o tasklist.o
25: MONITOR_APP_OBJS=tm_monitor.o
26:
27: FLTFLAGS+=-s 2097152
28: export FLTFLAGS
29:
30: LDLIBS += -lSDM -lpthread -lstdc++
31: LDFLAGS += -L../common/
32: all: $(APP) $(MONITOR_APP)
33:
34: $(APP): $(APP_OBJS)
35: $(CXX) $(LDFLAGS) -o $@ $(APP_OBJS) $(LDLIBS)
36: cp $@ ../FLIGHT_BINARIES/
37:
38: $(MONITOR_APP): $(MONITOR_APP_OBJS)
39: $(CXX) $(LDFLAGS) -o $@ $(MONITOR_APP_OBJS) $(LDLIBS)
```

```

40:
41: clean:
42: -rm -f $(APP) $(MONITOR_APP) *.elf *.gdb *.o
43:
44: romfs:
45: $(ROMFSINST) $(APP) /bin/$(APP)
46: $(ROMFSINST) $(MONITOR_APP) /bin/$(MONITOR_APP)
47: $(ROMFSINST) SdmTaskList.config /bin/SdmTaskList.config
48:
49: TMSUtil.o: ../dm/DMUtils.cpp ../dm/DMUtils.h
50: $(CXX) -c $(CXXFLAGS) -o $$@ $$<
51:
52: %.o: %.cpp
53: $(CXX) -c $(CXXFLAGS) -o $$@ $$<
54:
55: # Targets for the required .config files - if they don't exist, the tree isn't
56: # configured. Tell the user this, how to fix it, and exit.
57: ${ROOTDIR}/config.arch ${ROOTDIR}/.config:
58: @echo "Error: You must configure the PetaLinux tree before compiling your application"
59: @echo ""
60: @echo "Change directory to ../../petalinux-dist and 'make menuconfig' or 'make xconfig'"
61: @echo ""
62: @echo "Once the tree is configured, return to this directory, and re-run make."
63: @echo ""
64: @exit -1
65:

```

File: sdm/tm/Makefile

```
1: # Makefile for tm system
2: include ../Makefile.common
3: include ../$(MAKEFILE_DEFS)
4:
5: .PHONY:    clean distclean
6:
7: TMOBJECTS=tm.o pm_record.o pm_record_list.o TMUtils.o task.o tasklist.o
8:
9: all:  tm tm_monitor
10:
11: tm: $(TMOBJECTS)
12: $(CXX) $(CXXFLAGS) -L../common -static -o $$@ $$^ $(BOOSTFLAGS) -lSDM -lpthread
13:
14: tm_monitor:tm_monitor.cpp
15: $(CXX) $(CXXFLAGS) -L../common -static -o $$@ $$< $(BOOSTFLAGS) -lSDM -lpthread
16:
17: tm.o:  tm.cpp ../common/message_defs.h task.h tasklist.h ../common/checksum/checksum.h
18: $(CXX) $(CXXFLAGS) -c $$<
19:
20: pm_record.o:  pm_record.cpp pm_record.h
21: $(CXX) $(CXXFLAGS) -c $$<
22:
23: pm_record_list.o: pm_record_list.cpp pm_record_list.h
24: $(CXX) $(CXXFLAGS) -c $$<
25:
26: TMUtils.o: ../dm/DMUtils.cpp ../dm/DMUtils.h
27: $(CXX) $(CXXFLAGS) -c -o $$@ $$<
28:
29: clean:
30: rm -f *.o *~ SDMMessages*
31:
32: distclean: clean
33: rm -f tm tm_monitor
```

File: sdm/tm/tasklist.h

```
1: // TaskList definition file
2: #ifndef __TASKLIST_H_
3: #define __TASKLIST_H_
4:
5: #include "task.h"
6: #include "TmXtedsDefs.h"
7:
8: class TaskList
9: {
10: public:
11: TaskList();
12: void Init();
13: bool AnyInactive();
14: bool AnyPending();
15: bool AddTo(const Task& t);
16: bool RemoveFrom(unsigned int pid);
17: bool RemoveFrom(char *filename);
18: bool TaskFinished(unsigned int a_uiPid);
19: bool IsFilePresent(char *filename);
20: bool SetAddress(unsigned int pid, const SDMComponent_ID& PM);
21: unsigned int SetState (unsigned int pid, char state);
22: int FindPendingTask(Task& taskOut);
23: bool FindPendingMatch (int resources, unsigned int newPID, Task& taskOut);
24: unsigned int FindPID(const char* filename);
25: SDMComponent_ID GetPMAAddress(unsigned int pid);
26: void ClearList();
27: void PMFailure(const SDMComponent_ID& PMID);
28: void RemovePreferredPmId(const SDMTaskResources& TaskResources);
29: unsigned int FillTaskListBlob (char* pBuffer, unsigned int uiBufferSize);
30: unsigned int FillTaskListBlobRunningOnly (char* pBuffer, unsigned int uiBufferSize);
31: unsigned int FillTaskInfoRequest(unsigned int uiPid, char* pBuffer, unsigned int uiBufferSize);
32: void PrintList();
33: Task& operator[](int index);
34: private:
35: enum TaskSet { TASKS_ALL, TASKS_RUNNING };
36: unsigned int FillTaskListBlob (char* pBuffer, unsigned int uiBufferSize, TaskSet eTaskSet);
37: Task tasks[NUMTASKS];
38: };
39:
```

40: #endif

File: sdm/tm/pm_record.cpp

```
1: #include "pm_record.h"
2:
3: pm_record::pm_record():component_id(),resources(0),tasks(0),in_use(false)
4: {
5: }
6:
7: pm_record::pm_record(const pm_record &
b):component_id(b.component_id),resources(b.resources),tasks(b.tasks),in_use(b.in_use)
8: {
9: }
10: pm_record::~~pm_record()
11: {
12: }
13:
14: pm_record& pm_record::operator= (const pm_record& b)
15: {
16: resources = b.resources;
17: tasks = b.tasks;
18: in_use = b.in_use;
19: component_id = b.component_id;
20: return *this;
21: }
```

File: sdm/tm/tm.cpp

```
1: //*****
2: //Task Manager Module of the SDM
3: //
4: //This source is compilable on both a Linux and a Win32 architecture. Throughout, there are
5: //compile switches as #ifdef ... #endif which conditionally add or remove functionality.
6: //The most notable changes for compilation is the removal of a separate signal handling thread
7: //under Win32, because Windows has far fewer options for process signals than Linux.
8: //*****
9: #include "tm.h"
10: #include <list>
11: #include <algorithm>
12:
13: #ifndef WIN32
14: # include <net/if.h>
15: # include <netdb.h>
16: #endif
17:
18: #ifdef __VXWORKS__
19: #include <hostLib.h>
20: #include <pipeDrv.h>
21: #endif
22:
23: int debug;                                //level of debug info to output
24: bool spacewire = false;
25: unsigned char g_ucMode = 0;               //meaning TBD
26:
27: #ifdef __uLinux__
28: #define PM_HEARTBEAT_TIMEOUT (60000)
29: #else
30: #define PM_HEARTBEAT_TIMEOUT (5000)
31: #endif
32:
33: #ifndef __VXWORKS__
34: int pm_heartbeat_pipe[2];
35: #else
36: int pm_heartbeat_pipe;
37: #endif
38:
39: #ifdef PNP_BACKUP                          //Pipe for heartbeat messages
```



```

40: int tm_main_heartbeat_pipe[2]; //Pipe for heartbeat messages
41: int tm_back_heartbeat_pipe[2]; //Pipe for heartbeat messages
42: #endif
43:
44: #ifdef __uClinux__
45: const char* FILE_SDM_TASK_LIST = "/mnt/flash/etc/SdmTaskList.config";
46: #else
47: const char* FILE_SDM_TASK_LIST = "SdmTaskList.config";
48: #endif
49:
50: TaskList taskList; //the list of pending tasks
51: SubscriptionManager subscriptions; //a subscription manager to handle data publication
52: PMRecordList pmList; //a list of available process managers
53: MessageLogger log_service;
54:
55: pthread_mutex_t pid_mutex = PTHREAD_MUTEX_INITIALIZER;
56: static long NextPID = 1; // next available taskid number
57:
58: unsigned long Address_TM = ADDR_LOCAL_HOST; //TM IP address
59: unsigned int total_rcd = 0; //message counter for total received for life of tm
60: unsigned int prevsec_rcd = 0; //message counter for total received previous second
61: unsigned int total_sent = 0; //message counter for total sent for life of tm
62: unsigned int prevsec_sent = 0; //message counter for total sent previous second
63:
64: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER; //a mutex for the
subscription manager
65: pthread_mutex_t list_mutex = PTHREAD_MUTEX_INITIALIZER; //a mutex for the
task list
66: pthread_mutex_t mode_mutex = PTHREAD_MUTEX_INITIALIZER; //a mutex for
mode
67: pthread_mutex_t perf_counter_mutex = PTHREAD_MUTEX_INITIALIZER; //a mutex for
performance counter access permission
68: pthread_mutex_t log_service_mutex = PTHREAD_MUTEX_INITIALIZER; //a mutex for the
log service
69: pthread_mutex_t dm_found_mutex = PTHREAD_MUTEX_INITIALIZER; //a mutex for a
dm_found variable
70: #ifdef WIN32
71: pthread_mutex_t code_transfer_mutex = PTHREAD_MUTEX_INITIALIZER; //a mutex for
code transfers under windows
72: #endif
73:
74: #ifdef PNP_BACKUP

```

```

75: bool isBackup = true;           // defaults to true, you aren't the main until the NM tells you so
76: list<SDMComponent_ID> Backup_TM_List;    // list of addresses for the backups
77: #endif
78: SDMComponent_ID mainTM;           // main TM IP address
79: bool use_nm = false;
80: SDMComponent_ID NetworkManager;
81: unsigned long NM_Address;
82:
83: bool g_bSchedulerReset = false;
84: pthread_mutex_t g_mutexSchedulerReset = PTHREAD_MUTEX_INITIALIZER;
85:
86:
87: #ifdef BUILD_FOR_PNPSAT
88: //
89: // PowerController data items
90: bool g_bPowerControllerFound = false;
91: const unsigned short REQREG_POWER_CONTROLLER_ID = 10;
92: MessageManipulator g_mmEndpointReset;
93: SDMComponent_ID g_cidPowerController;
94: SDMMMessage_ID g_midEndpointReset;
95: pthread_mutex_t g_mutexPowerControllerData = PTHREAD_MUTEX_INITIALIZER;
96:
97: #endif // BUILD_FOR_PNPSAT
98:
99: bool ackReceived = false;
100: bool registerReceived = false;
101: bool idReceived = false;
102:
103:
104: bool dm_found = false;           //determines whether the dm has been found
105: PMRecordList pending_pms;       //vector of pending PM registrations
106: #define THREAD_STACK_SIZE 256000
107: int main(int argc, char** argv)
108: {
109:   SDM_TimeInit();
110:   pthread_t ListenerThread;
111:   pthread_t SchedulerThread;
112:   pthread_t SigHandlerThread;
113: #ifdef PNP_BACKUP
114:   pthread_t TMHeartbeatThread;
115: #endif

```

```

116: pthread_t PMHeartbeatThread;
117: bool DM_set = false;
118: bool bDoInitialStartup = true;
119:
120:
121: #ifndef __VXWORKS__
122:     if((Address_TM = GetNodeAddress()) == 0)
123: #else
124:     char hostName[50];
125:     int hostNameLength = 50;
126:     gethostname(hostName, hostNameLength);
127:     if((Address_TM = hostGetByName(hostName)) == 0)
128: #endif
129:     {
130:         printf("Unable to get the TM's IP address, using localhost instead. \n");
131:         Address_TM = ADDR_LOCAL_HOST;
132:     }
133: else
134:     {
135:         debug_f(3, "Task Manager address is 0x%lx \n", Address_TM);
136:     }
137:
138: // Set the Task Manager's node address
139: TaskManager.setAddress(Address_TM);
140: TaskManager.setPort(PORT_TM);
141:
142: // parse command line options
143: while(1)
144:     {
145: #ifndef __VXWORKS__
146:         static struct option long_options[] = {
147:             {"dm", 1, 0, 'd'},
148:             {"nm", 1, 0, 'n'},
149:             {"help", 0, 0, 'h'},
150:             {"debug", 1, 0, 'g'},
151:             {"spacewire", 1, 0, 's'},
152:             {"faulted", 0, 0, 'f'},
153:             {"config", 0, 0, 'c'},
154:             {0, 0, 0, 0} };
155:         int option_index;
156:         int option = getopt_long(argc, argv, "d:n:hg:s:fc:", long_options, &option_index);

```

```

157: #else
158:     int option = getopt(argc,argv,"d:n:hg:s:fc:");
159: #endif
160:     if(option== -1)
161:         break; // no more options
162:     switch(option)
163:     {
164:         case 'h':
165:             printf("Usage: tm [options] \n");
166:             printf("Options: \n");
167:             printf("--dm=<addr> -d<addr> \t \t \tSet IP address of Data Manager \n \t \t \t \t
\t<addr> should be given in dot number notation \n");
168:             printf("--nm=<addr> -n<addr> \t \t \tSet IP address of Network Manager \n \t \t \t \t
\t<addr> should be given in dot number notation \n");
169:             printf("--help -h \t \t \tDisplay this information \n");
170:             printf("--spacewire=<bool> -s<bool> \t \tEnable or disable SpaceWire mode \n");
171:             printf("--debug=<level> -g<level> \t \tSet level of debug messages \n \t \t \t \t
\t0=none, 1=moderate, 2=verbose \n");
172:             printf("--config=<configFile> -c<configFile> \t \tName of config file \n \t \t \t \t
\tlocal name or full path to SdmTaskList.config equivalent \n");
173:             return 0;
174:
175:         case 'd':
176:             DataManager.setAddress(inet_addr(optarg));
177:             if (((long)(DataManager.getAddress())) == -1)
178:             {
179:                 printf("Error in DM address. Be sure to use --dm= instead of -dm= \n");
180:                 return 0;
181:             }
182:             DataManager.setPort(PORT_DM);
183:             DM_set = true;
184:             // note, this can be overridden by --spacewire
185:             break;
186:
187:         case 'n':
188:             use_nm = true;
189:             NM_Address = inet_addr(optarg);
190:             break;
191:
192:         case 'g':
193:             debug = atoi(optarg);
194:             break;

```

```

195:
196:     case 's':
197:         spacewire = !!atoi(optarg);
198:         break;
199:
200:     case 'f':
201:         // The restart is a result of a fault, don't load tasks
202:         bDoInitialStartup = false;
203:         break;
204:
205:     case 'c':
206:         FILE_SDM_TASK_LIST = strdup(optarg);
207:         break;
208:
209:     case '?':
210:         break;
211: }
212: }
213: #ifdef PNP_BACKUP
214: // request the DM's address from the NM
215: if (use_nm)
216: {
217:     NetworkManager.setAddress (NM_Address);
218:     NetworkManager.setPort(1982);
219: }
220: else
221: {
222:     NetworkManager.setAddress ( inet_addr("127.0.0.1") );
223:     NetworkManager.setPort(1982);
224: }
225: #endif
226: #ifdef SEND_IMA
227: if (spacewire)
228: {
229:     debug_f(1, "Running in SpaceWire mode \n");
230:     SendIMA(ImaTm, debug);
231: }
232: #endif
233: #ifndef __VXWORKS__
234: if ((Address_TM = GetNodeAddress(spacewire)) == 0)
235: #else

```

```

236: //TODO: The GetNodeAddress(bool) function call is causing problems
237: //This work around probably is not sufficient.
238: gethostname(hostName, hostNameLength);
239: if((Address_TM = hostGetByName(hostName)) == 0)
240: #endif
241: {
242:     debug_f(0, "Unable to get the TM's IP address, using localhost instead. \n");
243:     Address_TM = ADDR_LOCAL_HOST;
244: }
245: else
246: {
247:     debug_f(3, "Task Manager address is 0x%lx \n",Address_TM);
248: }
249:
250: // Set the Task Manager's node address
251: TaskManager.setAddress(Address_TM);
252: TaskManager.setPort(PORT_TM);
253:
254: #ifdef PNP_FAKE
255: MessageManager mm;
256: mm.Async_Init_Both(PORT_TM);
257:
258: AmIBackup(&mm);
259: #endif
260:
261: #ifdef PNP_BACKUP
262: isBackup = AmIBackup();
263: #endif
264:
265: // If the DM's address was not specified on the command line, assume local
266: if(!DM_set && !spacewire)
267: {
268:     DataManager.setAddress(inet_addr("127.0.0.1"));
269:     DataManager.setPort(PORT_DM);
270: }
271: else if(spacewire)
272: {
273:     long addr;
274: #ifndef __VXWORKS__
275:     struct hostent *he;
276:     while ((he=gethostbyname("datamanager.spacewire")) == NULL)

```

```

277:     {
278:         sleep(1);
279:     }
280:     memcpy(&addr, he->h_addr, sizeof(addr));
281: #else
282:     while ((addr = hostGetByName("datamanager.spacewire")) == ERROR)
283:     {
284:         sleep(1);
285:     }
286: #endif
287:
288:     DataManager.setAddress(addr);
289:     DataManager.setPort(PORT_DM);
290:     DM_set = true;
291: }
292:
293: #ifdef PNP_FAKE
294: RequestDMInfo(&mm);
295: #endif
296:
297: #ifdef PNP_BACKUP
298: RequestDMInfo();
299: #endif
300:
301: // output version and debug level information
302: printf("TM (Task Manager) %s \n",SDM_VERSION);
303: switch(debug)
304: {
305:     case 1:
306:         printf("TM in debug 1 (moderate). \n");
307:         break;
308:     case 2:
309:         printf("TM in debug 2 (verbose). \n");
310:         break;
311:     case 3:
312:         printf("TM in debug 3 (verbose w/message echo). \n");
313:         break;
314:     default:
315:         break;
316: }
317:

```

```

318: pthread_attr_t threadAttr;
319: pthread_attr_init(&threadAttr);
320: pthread_attr_setstacksize(&threadAttr, THREAD_STACK_SIZE);
321:
322: // All subsequent threads block the SIGINT and SIGALRM signals so they aren't interrupted after
calling pthread_mutex_lock()
323: // this avoids a deadlock situation by dedicating a single thread for signal handling.
324: #ifndef WIN32
325: sigset_t signal_set;
326: sigemptyset(&signal_set);
327: sigaddset(&signal_set, SIGALRM);
328: sigaddset(&signal_set, SIGINT);
329: pthread_sigmask(SIG_BLOCK, &signal_set, NULL);
330:
331: if (0 != pthread_create(&SigHandlerThread, &threadAttr, &SigHandler, NULL))
332: {
333:     perror("Could not create signal handler thread. \n");
334:     return -1;
335: }
336:
337: // Set up pipe for the heartbeat thread to receive heartbeat responses
338: #ifndef __VXWORKS__
339: pipe(pm_heartbeat_pipe);
340: #else
341: if(pipeDevCreate("/pipe/pmHeartbeat", 10, 100) != 0)
342: {
343:     if(debug >= 2)
344:         fprintf(stderr, "Pipe error \n");
345: }
346: pm_heartbeat_pipe = open("/pipe/pmHeartbeat", O_RDWR);
347: #endif
348:
349:
350: #ifdef PNP_BACKUP
351: pipe(tm_back_heartbeat_pipe);
352: pipe(tm_main_heartbeat_pipe);
353: if (0 != pthread_create(&TMHeartbeatThread, &threadAttr, &TMHeartbeat, NULL))
354: {
355:     perror("Could not create TM heartbeat thread. \n");
356:     return -1;
357: }

```



```

358: #endif
359:
360:     if (0 != pthread_create(&PMHeartbeatThread, &threadAttr, &PMHeartbeat, NULL))
361:     {
362:         perror("Could not create PM heartbeat thread. \n");
363:         return -1;
364:     }
365:
366: #else
367:     // Windows signal handling
368:     signal(SIGINT, SigHandler);
369:     sigset(SIGALRM, SigHandler);
370: #endif
371:     if (bDoInitialStartup)
372:     {
373:         InitialStartUp();
374:     }
375:
376:     if (0 != pthread_create(&SchedulerThread, &threadAttr, &Scheduler, NULL))
377:     {
378:         perror("Could not create scheduler thread. \n");
379:         return -1;
380:     }
381:
382: #ifdef PNP_FAKE
383:     if (0 != pthread_create(&ListenerThread, &threadAttr, &Listener, &mm))
384:     {
385:         perror("Could not create Listen thread. \n");
386:         return -1;
387:     }
388: #else
389:     if (0 != pthread_create(&ListenerThread, &threadAttr, &Listener, NULL))
390:     {
391:         perror("Could not create Listen thread. \n");
392:         return -1;
393:     }
394: #endif
395:     // verify that DM is present
396:     VerifyDM();
397:
398:     debug_f(2, "Finding DM at address 0x%x\n", DataManager.getAddress());

```

```

399:   debug_f(2,"TM address is 0x%lx \n",Address_TM);
400:
401: #ifdef PNP_BACKUP
402:  // at this point we should know who is leader and backup
403:  if (!isBackup)
404:  {
405: #endif
406:   SDMHHello hello;
407:   hello.source.setPort(PORT_TM);
408:   hello.type = 'C';
409:   double endTime = 0;
410:   double timeOut = 5.0;
411:   while(!ackReceived)
412:   {
413:     if(GetCurTime() > endTime)
414:     {
415:       debug_f(1, "Sending Hello \n");
416:       hello.Send();
417:       endTime = GetCurTime() + timeOut;
418:     }
419:     usleep(10000);
420:   }
421:   while(!registerReceived)
422:   {
423:     usleep(10000);
424:   }
425:   debug_f(1, "Registering xTEDS \n");
426:
427:
428:
429:
430:   // post xTEDS to the DM
431:   SDMxTEDS msgXteds;
432:   msgXteds.source.setSensorID(1);
433:   msgXteds.source.setPort(PORT_TM);
434:   strcpy(msgXteds.xTEDS, STR_TASK_MANAGER_XTEDS);
435:   msgXteds.Send();
436:   MessageSent(&msgXteds);
437:
438:   while (!lidReceived)
439:   {

```

```

440:     usleep(10000);
441: }
442:     debug_f(1, "SDMID received \n");
443:
444: #ifdef PNP_BACKUP
445: }
446: #endif
447: #ifdef BUILD_FOR_PNPSAT
448: //
449: // If this is a build for PnPsat, request the PowerController's interface for
450: // cycling power to an endpoint
451:     RequestRegPowerController();
452: #endif // BUILD_FOR_PNPSAT
453:
454: #ifndef WIN32
455: #ifdef PNP_BACKUP
456:     pthread_join(TMHeartbeatThread, NULL);
457: #endif
458:     pthread_join(PMHeartbeatThread, NULL);
459:     pthread_join(SigHandlerThread, NULL);
460: #endif
461:     pthread_join(ListenerThread, NULL);
462:     pthread_join(SchedulerThread, NULL);
463: }
464:
465:
466:
467: double GetCurTime()
468: {
469:     unsigned int seconds;
470:     unsigned int uSeconds;
471:     double curTime;
472:     SDM_GetTime(&seconds, &uSeconds);
473:     curTime = seconds + ((double)uSeconds/1000000.0);
474:     return curTime;
475: }
476:
477:
478:
479: /*

```

```

480: * Separate thread for signal handling. This avoids a potential deadlock situation in which the
ChildFunctionCallProcess function
481:     * is chosen to handle the SIGALRM signal after it has called
pthread_mutex_lock(&subscription_mutex), which is also called in the
482: * signal handler.
483: */
484: #ifndef WIN32
485: void* SigHandler(void* arg)
486: {
487:     SDMCancelxTEDS msgCancel;
488:     sigset_t signal_set;
489:     int sig;
490:     msgCancel.source.setSensorID(1);
491:     msgCancel.source.setPort(PORT_TM);
492:     sigemptyset(&signal_set);
493:     sigaddset(&signal_set, SIGINT);
494:     sigaddset(&signal_set, SIGALRM);
495:     sigaddset(&signal_set, SIGSEGV);
496:
497:     while (1)
498:     {
499:         sigwait(&signal_set, &sig);
500:         switch (sig)
501:         {
502:             case SIGINT:
503:                 printf(" \nShutting down \n");
504:                 pthread_mutex_lock(&dm_found_mutex);
505: #ifdef PNP_BACKUP
506:                 if (dm_found && !isBackup && Backup_TM_List.empty()) // Cancel the TM's xTEDS
507: #endif
508:                     msgCancel.Send();
509:                     pthread_mutex_unlock(&dm_found_mutex);
510:                     MessageSent(&msgCancel);
511:                     exit(EXIT_SUCCESS);
512:                     break;
513:             case SIGALRM:
514:                 PublishPerformanceCounterMessage();
515:                 pthread_mutex_lock(&perf_counter_mutex);
516:                 prevsec_recd = 0;
517:                 prevsec_sent = 0;
518:                 pthread_mutex_unlock(&perf_counter_mutex);

```

```

519:         break;
520:     default:
521:         printf("Unexpected signal received: %i \n", sig);
522:         break;
523:     }
524: }
525: }
526: #else
527: /*Win32 version uses a SigHandler routine that handles only SIGINT.*/
528: void SigHandler(int signum)
529: {
530:     SDMCancelxTEDS msgCancel;
531:     msgCancel.source.setSensorID(1);
532:     msgCancel.source.setPort(PORT_TM);
533:
534:     if (signum == SIGINT)
535:     {
536:         printf(" \nShutting down \n");
537:         pthread_mutex_lock (&dm_found_mutex);
538:         if (dm_found)
539:             msgCancel.Send();
540:         pthread_mutex_unlock (&dm_found_mutex);
541:         MessageSent(&msgCancel);
542:         exit (EXIT_SUCCESS);
543:     }
544:     else if ( signum == SIGALRM )
545:     {
546:         PublishPerformanceCounterMessage();
547:         pthread_mutex_lock(&perf_counter_mutex);
548:         prevsec_recd = 0;
549:         prevsec_sent = 0;
550:         pthread_mutex_unlock(&perf_counter_mutex);
551:     }
552: }
553: #endif
554: /*
555:     PublishStatusMessage publishes the TaskManager's status message to all subscribers. This status
556:     message is used for (possible among other
557:     things), notifying all SDM components if a DataManager has failed and a new one becomes
    elected.
557:     INPUTS:

```

```

558:     sender_buf - Contains the data to be sent in the mode changes message (new DM IP address)
559: RETURNS:
560:     void
561: */
562: void PublishStatusMessage(char *sender_buf)
563: {
564:     char buf[11];
565:     // publish status message
566:     debug_f(2,"Publishing Status \n");
567:
568:     pthread_mutex_lock(&mode_mutex);
569:     buf[0] = g_ucMode;
570:     pthread_mutex_unlock(&mode_mutex);
571:
572:     if (sender_buf == NULL)
573:         memset(&buf[1],0,10);
574:     else
575:         memcpy(&buf[1],sender_buf,10);
576:
577:     pthread_mutex_lock(&subscription_mutex);
578:     if(subscriptions.Publish(NOTI_TM_STATUS,buf,11))
579:     {
580:         MessageSent(subscriptions.GetLastPublished());
581:         debug_f(3,"Status Message Sent \n");
582:     }
583:     else
584:         debug_f(3,"No Status Message Subscribers \n");
585:     pthread_mutex_unlock(&subscription_mutex);
586: }
587: /*
588: PublishTaskQueuedMessage publishes to all subscribers, the name of a newly queued task for
execution. Any time a new task is posted
589: to the TaskManager, it publishes this message.
590: INPUTS:
591:     taskname - The name of the task posted.
592: RETURNS:
593:     void
594: */
595: void PublishTaskQueuedMessage(char* taskname)
596: {
597:     debug_f(2,"Publishing TaskQueued message for taskname \"%s\" \n", taskname);

```

```

598:
599:  char filename[MAX_FILENAME_SIZE];
600:  memset(filename, '\0', sizeof(filename));
601:  strncpy(filename, taskname, sizeof(filename));
602:  filename[sizeof(filename) - 1] = '\0';
603:
604:  pthread_mutex_lock(&subscription_mutex);
605:  if(subscriptions.Publish(NOTI_TASK_QUEUED, filename, sizeof(filename)))
606:  {
607:      MessageSent(subscriptions.GetLastPublished());
608:      debug_f(3,"TaskQueued Message Sent \n");
609:  }
610:  else
611:      debug_f(3,"No TaskQueued Message Subscribers \n");
612:
613:  pthread_mutex_unlock(&subscription_mutex);
614: }
615: /*
616:  PublishTaskStartedMessage publishes to all subscribers, the name of any task that gets posted to
  a ProcessManager for execution.
617:  This message is published any time a task leaves the queued state and starts the running state.
618:  INPUTS:
619:      taskname - The name of the task started.
620:  RETURNS:
621:      void
622:  */
623: void PublishTaskStartedMessage(char* taskname, unsigned int uiPid)
624: {
625:  debug_f(2,"Publishing TaskStarted message for taskname \"%s\" pid=%u \n", taskname, uiPid);
626:  const unsigned int PUBLISH_BUFFER_SIZE = XTEDS_MAX_TASK_NAME_SIZE +
  sizeof(unsigned int);
627:
628:  char PublishBuffer[PUBLISH_BUFFER_SIZE];
629:  memset(PublishBuffer, '\0', sizeof(PublishBuffer));
630:  strncpy(PublishBuffer, taskname, XTEDS_MAX_TASK_NAME_SIZE);
631:  PublishBuffer[XTEDS_MAX_TASK_NAME_SIZE - 1] = '\0';
632:
633:  PUT_UINT(PublishBuffer + XTEDS_MAX_TASK_NAME_SIZE, uiPid);
634:
635:  pthread_mutex_lock(&subscription_mutex);
636:  if(subscriptions.Publish(NOTI_TASK_STARTED, PublishBuffer, PUBLISH_BUFFER_SIZE))

```

```

637:  {
638:      MessageSent(subscriptions.GetLastPublished());
639:      debug_f(3,"TaskStarted Message Sent \n");
640:  }
641:  else
642:      debug_f(3,"No TaskStarted Message Subscribers \n");
643:
644:  pthread_mutex_unlock(&subscription_mutex);
645: }
646: /*
647:  PublishTaskFinishedMessage publishes to all subscribers the name of any task that finishes
  execution on a ProcessManager.
648:  This message is published when a ProcessManager node informs the TaskManager that a task has
  completed.
649:  INPUTS:
650:      taskname - The name of the task that has finished.
651:      exitcode - The exit code status of the task that finished.
652:  RETURNS:
653:      void
654:  */
655: void PublishTaskFinishedMessage(char* taskname,int exitcode, unsigned int uiPid)
656: {
657:     debug_f(2,"Publishing TaskFinished message for taskname  \"%s \", exit code %d \n", taskname,
  exitcode);
658:     const unsigned int PUBLISH_BUFFER_SIZE = XTEDS_MAX_TASK_NAME_SIZE +
659:         sizeof(int) + sizeof(unsigned int);
660:
661:     char PublishBuffer[PUBLISH_BUFFER_SIZE];
662:     memset(PublishBuffer, '\0', sizeof(PublishBuffer));
663:     strncpy(PublishBuffer, taskname, XTEDS_MAX_TASK_NAME_SIZE);
664:     PublishBuffer[XTEDS_MAX_TASK_NAME_SIZE - 1] = '\0';
665:
666:     PUT_UINT(PublishBuffer + XTEDS_MAX_TASK_NAME_SIZE, uiPid);
667:
668:     PUT_INT(PublishBuffer + XTEDS_MAX_TASK_NAME_SIZE + sizeof(unsigned int),
  exitcode);
669:
670:     pthread_mutex_lock(&subscription_mutex);
671:     if(subscriptions.Publish(NOTI_TASK_FINISHED,PublishBuffer,PUBLISH_BUFFER_SIZE))
672:     {
673:         MessageSent(subscriptions.GetLastPublished());
674:         debug_f(3,"TaskFinished Message Sent \n");

```



```

675:     }
676:     else
677:         debug_f(3,"No TaskFinished Message Subscribers \n");
678:
679:     pthread_mutex_unlock(&subscription_mutex);
680: }
681: /*
682:     PublishPerformanceCounterMessage publishes the TaskManager's message performance counters
to all subscribers every second if there has
683:     been a subscription request to the performance counter messages. Otherwise, the system time has
not been started and the performance
684:     counters are not published.
685:     INPUTS:
686:         None.
687:     RETURNS:
688:         void
689: */
690: void PublishPerformanceCounterMessage(void)
691: {
692:     debug_f(2,"Publishing Message_Count \n");
693:
694:     char msg[16];
695:     pthread_mutex_lock(&perf_counter_mutex);
696:     PUT_UINT(&msg[0], total_recd);
697:     PUT_UINT(&msg[4], prevsec_recd);
698:     PUT_UINT(&msg[8], total_sent);
699:     PUT_UINT(&msg[12], prevsec_sent);
700:     pthread_mutex_unlock(&perf_counter_mutex);
701:
702:     pthread_mutex_lock(&subscription_mutex);
703:     if (subscriptions.Publish(NOTI_PERF_COUNTERS, msg, 16))
704:     {
705:         MessageSent(subscriptions.GetLastPublished());
706:         debug_f(3,"Message_Count Message Sent \n");
707:     }
708:     else
709:         debug_f(3,"Message_Count Message Subscribers \n");
710:
711:     pthread_mutex_unlock(&subscription_mutex);
712: }
713: /*

```

```

714:   RegPMHandler handles the registration of a ProcessManager to the system. This routine stores
the IP address and node resource
715:   limits in the pmList for scheduling.
716:   INPUTS:
717:       arg - The buffer containing the SDMRegPM message that was received.
718:   RETURNS:
719:       void * - Always NULL.
720: */
721: void* RegPMHandler(void* arg)
722: {
723:     char* buf = (char*)arg;
724:     SDMRegPM msgReg;
725:
726:     if (msgReg.Unmarshal(buf) < 0)
727:     {
728:         printf("Received invalid SDMRegPM message. \n");
729:         return NULL;
730:     }
731:     else
732:     {
733:         MessageReceived(&msgReg);
734:     }
735:
736:     debug_f(1,"PM registering at ip 0x%lx \n", msgReg.source.getAddress());
737:
738:     // Check to see if the PM node is already registered with the TM
739:     if (pmList.AlreadyRegistered(msgReg.source, msgReg.resources))
740:     {
741:         // duplicate machine there can be only one pm per ip address
742:         debug_f(1, "PM node already registered. \n");
743:         if (debug >= 3)
744:             pmList.PrintList();
745:         return NULL;
746:     }
747:
748:     if (!pmList.RegisterPM(msgReg.source, msgReg.resources))
749:     {
750:         debug_f(0,"Error registering PM, PM list full. \n");
751:     }
752:     // See if there is an entry in pending_pms for this ProcessManager if so, copy its task number
753:     if (pending_pms.AlreadyRegistered(msgReg.source))

```

```

754:  {
755:      pmList.SetTasks(msgReg.source, pending_pms.GetTasks(msgReg.source));
756:  }
757:
758:  if (debug >= 3)
759:  {
760:      pmList.PrintList();
761:  }
762:
763:  return NULL;
764: }
765: /*
766:  ReqCodeHandler handles the receipt of an SDMCode message.  This function calls SendCode
  directly.
767:  INPUTS:
768:      arg - The buffer containing the SDMCode message.
769:  RETURNS:
770:      void * - Always NULL.
771: */
772: void* ReqCodeHandler(void* arg)
773: {
774:     char* buf = (char*)arg;
775:     debug_f(1,"Request for code \n ");
776:     SendCode (buf);
777:     delete[] buf;
778:     return NULL;
779: }
780: /*
781:  PostTaskHandler handles the receipt of an SDMTask message, to post a task to a
  ProcessManager.
782:  This function calls PostTask directly.
783:  INPUT:
784:      arg - The buffer containing the SDMTask message.
785:  RETURNS:
786:      void * - Always NULL.
787: */
788: void* PostTaskHandler(void* arg)
789: {
790:     char* buf = (char*)arg;
791:     debug_f(1,"Post task \n ");
792:     PostTask (&buf[0]);

```

```

793:   return NULL;
794: }
795: /*
796:   SubreqstHandler handles the receipt of an SDMSubreqst message. This routine gets the
subscription request's message and
797:   interface ID's and adds the request to the TM's subscription list.
798:   INPUTS:
799:       arg - The buffer containing the SDMSubreqst message.
800:   RETURNS:
801:       void * - Always NULL.
802: */
803: void* SubreqstHandler(void* arg)
804: {
805:   char* buf = (char*)arg;
806:   SDMSubreqst msgRequest;
807:
808:   // Unmarshal message
809:   if (msgRequest.Unmarshal(buf) < 0)
810:   {
811:     printf("Invalid SDMSubreqst message. \n");
812:     return NULL;
813:   }
814:   else
815:   {
816:     MessageReceived(&msgRequest);
817:   }
818:
819:   debug_f(1,"Subscription request (interface %hhd, message %hhd) \n",
msgRequest.msg_id.getInterface(), msgRequest.msg_id.getMessage());
820:
821:   // Add the subscription request to the TM's subscription list
822:   pthread_mutex_lock(&subscription_mutex);
823:   subscriptions.AddSubscription(msgRequest);
824:   pthread_mutex_unlock(&subscription_mutex);
825:
826: #ifdef PNP_BACKUP
827:   // Request for TM's status notification
828:   if (!isBackup && msgRequest.msg_id == NOTI_TM_STATUS)
829:   {
830:     PublishStatusMessage(NULL);
831:   }

```

```

832: #else
833:     // Request for TM's status notification
834:     if (msgRequest.msg_id == NOTI_TM_STATUS)
835:     {
836:         PublishStatusMessage(NULL);
837:     }
838: #endif
839:     else if (msgRequest.msg_id == NOTI_PERF_COUNTERS) // Request for TM's performance
counters
840:     {
841: #ifndef __VXWORKS__
842:         itimerval interval;
843:         getitimer(ITIMER_REAL, &interval);
844:         if (interval.it_value.tv_sec == 0 && interval.it_interval.tv_sec == 0)
845:         {
846:             // Time interval for the performance counter publish
847:             timeval pubInterval;
848:             pubInterval.tv_sec = 1;
849:             pubInterval.tv_usec = 0;
850:
851:             itimerval timerInterval;
852:             timerInterval.it_interval = pubInterval;
853:             timerInterval.it_value = pubInterval;
854:
855:             // Set the performance counter timer
856:             setitimer (ITIMER_REAL, &timerInterval, NULL);
857:         }
858: #else
859:         itimerspec interval;
860:         timer_gettime(CLOCK_REALTIME, &interval);
861:         if (interval.it_value.tv_sec == 0 && interval.it_interval.tv_sec == 0)
862:         {
863:             //Time interval for the publish interval of the performance counter
864:             itimerspec timerInterval;
865:             timerInterval.it_value.tv_sec = 1;
866:             timerInterval.it_value.tv_nsec = 0;
867:
868:             //Set the performance counter timer
869:             timer_settime(CLOCK_REALTIME, 0, &timerInterval, NULL);
870:         }
871: #endif

```

```

872:  }
873:
874:  return NULL;
875: }
876: /*
877:  DeletesubHandler handles the receipt of an SDMDeletesub message. This message removes the
specified subscription from the
878:  TaskManagers subscription list.
879:  INPUTS:
880:      arg - The buffer containing the SDMDeletesub message.
881:  RETURNS:
882:      void * - Always NULL.
883: */
884: void* DeletesubHandler(void* arg)
885: {
886:  char* buf = (char*)arg;
887:  SDMDeletesub msgDeleteSub;
888:
889:  // Unmarshal the message
890:  if (msgDeleteSub.Unmarshal(buf) < 0)
891:  {
892:      printf("Invalid SDMDeletesub message. \n");
893:      return NULL;
894:  }
895:  MessageReceived(&msgDeleteSub);
896:
897:  debug_f(1,"Subscription cancellation (interface %d, message %d) \n",
msgDeleteSub.msg_id.getInterface(), msgDeleteSub.msg_id.getMessage());
898:
899:  // Remove the subscription request from the TM's subscription list
900:  pthread_mutex_lock(&subscription_mutex);
901:  subscriptions.RemoveSubscription(msgDeleteSub);
902:  pthread_mutex_unlock(&subscription_mutex);
903:
904:  return NULL;
905: }
906: /*
907:  CommandHandler handles the receipt of an SDMCommand message. If the specified message
and interface IDs are appropriate for
908:  one of the TaskManager's command messages, the command is issued.
909:  INPUTS:
910:      arg - The buffer containing the SDMCommand message.

```

```

911: RETURNS:
912:     void * - Always NULL.
913: */
914: void* CommandHandler(void* arg)
915: {
916:     char* buf = (char*)arg;
917:     SDMCommand msgCommand;
918:
919:     // Unmarshal the message
920:     if (msgCommand.Unmarshal(buf) < 0)
921:     {
922:         printf("Invalid SDMCommand message. \n");
923:         delete [] buf;
924:         return NULL;
925:     }
926:     MessageReceived(&msgCommand);
927:
928:     debug_f(1,"Command request (interface %d, message %d)
\n",msgCommand.command_id.getInterface(), msgCommand.command_id.getMessage());
929:
930:     // Command to change the TM's mode
931:     if (msgCommand.command_id == CMD_CHANGE_MODE)
932:     {
933:         bool bDoHardReset = false;
934:         bool bDoSoftReset = false;
935:         unsigned char ucOldMode, ucModeCopy, ucNewMode;
936:         SDMComponent_ID idNewDm;
937:
938:         // Get the TM's current mode value
939:         pthread_mutex_lock(&mode_mutex);
940:         ucModeCopy = g_ucMode;
941:         pthread_mutex_unlock(&mode_mutex);
942:         ucNewMode = GET_UCHAR(&msgCommand.data[0]);
943:
944:         debug_f(2,"Command to change mode (TM current mode=%hhu, new mode=%hhu \n",
ucModeCopy, ucNewMode);
945:         // If the current mode is not the same as the new change received
946:         if (ucModeCopy != ucNewMode)
947:         {
948:             // change to new mode
949:             ucOldMode = ucModeCopy;

```

```

950:     pthread_mutex_lock(&mode_mutex);
951:     g_ucMode = ucNewMode;
952:     pthread_mutex_unlock(&mode_mutex);
953:
954:     if (ucNewMode == MODE_STATUS_HARD_RESET)
955:     {
956:         debug_f(1, "Performing a hard reset... \n");
957:         bDoHardReset = true;
958:     }
959:     else if (ucNewMode == MODE_STATUS_SOFT_RESET)
960:     {
961:         debug_f(1, "Performing a soft reset... \n");
962:         bDoSoftReset = true;
963:     }
964:     // Reset code for hard or soft grouped similarly
965:     // A soft reset should restart (kill) all of the tasks
966:     // running on each PM.
967:     // A hard reset needs to re-verify the DM and register
968:     // the usual xTEDS again.
969:     if (bDoHardReset || bDoSoftReset)
970:     {
971:         if (bDoHardReset)
972:         {
973:             // Reset dm_found, and re-verify the DM
974:             pthread_mutex_lock(&dm_found_mutex);
975:             dm_found = false;
976:             pthread_mutex_unlock(&dm_found_mutex);
977:             // Gets the new DM address
978: #ifdef PNP_BACKUP
979:             VerifyNewDM(&msgCommand.data[1]);
980:             idNewDm = DataManager;
981:             debug_f(2, "New DM address is 0x%lx:%hd %ld \n", idNewDm.getAddress(),
idNewDm.getPort(), idNewDm.getSensorID());
982: #endif
983:
984:
985:         }
986: #ifdef PNP_BACKUP
987:         if (!isBackup)
988:         {
989:             // Notify all PMs of the mode change

```



```

990:         PublishStatusMessage(&msgCommand.data[1]);
991:     }
992: #else
993:         PublishStatusMessage(&msgCommand.data[1]);
994: #endif
995:
996:     if (bDoHardReset)
997:     {
998:         VerifyDM();
999:         idNewDm = DataManager;
1000:         // The new DM address is acquired is responses to VerifyDM
1001:         debug_f(2,"New DM address is 0x%lx:%hd %ld \n",idNewDm.getAddress(),
idNewDm.getPort(), idNewDm.getSensorID());
1002:
1003:         // Reregister the TM's xTEDS
1004:         // SDMxTEDS msgXteds;
1005:         // msgXteds.source = TaskManager;
1006:         // msgXteds.source.setSensorID(1);
1007:         // strncpy(msgXteds.xTEDS, STR_TASK_MANAGER_XTEDS,
sizeof(msgXteds.xTEDS));
1008:         // msgXteds.Send();
1009:         // MessageSent(&msgXteds);
1010:     }
1011:     // Clear the task list
1012:     // Hold the list mutex as well to prevent the scheduler from restarting
1013:     // before the scheduler reset can be set and the task list from clearing
1014:     // This should be the only place that these two locks are held at the same
1015:     // time.
1016:     pthread_mutex_lock(&g_mutexSchedulerReset);
1017:     pthread_mutex_lock(&list_mutex);
1018:     g_bSchedulerReset = true;
1019:     taskList.ClearList();
1020:     pthread_mutex_unlock(&list_mutex);
1021:     pthread_mutex_unlock(&g_mutexSchedulerReset);
1022:
1023:     // Set all PM's to not having any running tasks
1024:     pmList.ClearAllRunningTasks();
1025:
1026:     // Re-Read the sdm.config file for any startup tasks
1027:     InitialStartUp();
1028:

```

```

1029:         // Reset the mode back to the mode before the reset
1030:         pthread_mutex_lock(&mode_mutex);
1031:         g_ucMode = ucOldMode;
1032:         pthread_mutex_unlock(&mode_mutex);
1033:
1034:         delete [] buf;
1035:         return NULL;
1036:     }
1037:     // If the mode change was not for a reset, publish here
1038: #ifdef PNP_BACKUP
1039:     if (!isBackup)
1040:     {
1041:         PublishStatusMessage(&msgCommand.data[1]);
1042:     }
1043: #else
1044:     PublishStatusMessage(&msgCommand.data[1]);
1045: #endif
1046:     }
1047: }
1048: // Command to enable message logging
1049: else if (msgCommand.command_id == CMD_ENABLE_LOGGING)
1050: {
1051:     pthread_mutex_lock(&log_service_mutex);
1052:     if (log_service.NeedsInit())
1053:         log_service.SetLogFile("Task Manager Message Log \n", "tmmessages.log");
1054:     log_service.AddMessageType(&msgCommand);
1055:     pthread_mutex_unlock(&log_service_mutex);
1056: }
1057: // Command to disable message logging
1058: else if (msgCommand.command_id == CMD_DISABLE_LOGGING)
1059: {
1060:     pthread_mutex_lock(&log_service_mutex);
1061:     log_service.RemoveMessageType(&msgCommand);
1062:     pthread_mutex_unlock(&log_service_mutex);
1063: }
1064: // Command to start a task
1065: else if (msgCommand.command_id == CMD_START_TASK)
1066: {
1067:     // Create a task request and post
1068:     char buf[256];
1069:     SDMPostTask msgTask;

```

```

1070:     msgTask.priority = 1;
1071:     unsigned int uiCurBufferOffset = 0;
1072:     SDMTaskResources taskResources;
1073:     //
1074:     // Get the task name
1075:     strncpy(msgTask.filename, msgCommand.data, sizeof(msgTask.filename));
1076:     msgTask.filename[sizeof(msgTask.filename) - 1] = '\0';
1077:     uiCurBufferOffset += XTEDS_MAX_TASK_NAME_SIZE;
1078:     //
1079:     // Get the arch type
1080:     char cArchType = GET_CHAR(&msgCommand.data[uiCurBufferOffset]);
1081:     switch( cArchType )
1082:     {
1083:     case ARCH_TYPE_INTEL:
1084:         taskResources.SetArch(SDM_INTEL);
1085:         break;
1086:     case ARCH_TYPE_MICROBLAZE:
1087:         taskResources.SetArch(SDM_MICROBLAZE);
1088:         break;
1089:     case ARCH_TYPE_SPARC:
1090:         taskResources.SetArch(SDM_SPARC);
1091:         break;
1092:     default:
1093:         break;
1094:     }
1095:     uiCurBufferOffset += sizeof(char);
1096:     //
1097:     // Get the OS type
1098:     char cOsType = GET_CHAR(&msgCommand.data[uiCurBufferOffset]);
1099:     switch( cOsType )
1100:     {
1101:     case OS_TYPE_LINUX26:
1102:         taskResources.SetOs(SDM_LINUX26);
1103:         break;
1104:     case OS_TYPE_WIN32:
1105:         taskResources.SetOs(SDM_WIN32);
1106:         break;
1107:     case OS_TYPE_VXWORKS:
1108:         taskResources.SetOs(SDM_VXWORKS);
1109:         break;
1110:     default:

```

```

1111:     break;
1112:     }
1113:     uiCurBufferOffset += sizeof(char);
1114:     //
1115:     // Get the mem type
1116:     char cMemType = GET_CHAR(&msgCommand.data[uiCurBufferOffset]);
1117:     switch( cMemType )
1118:     {
1119:     case MEM_TYPE_64:
1120:         taskResources.SetMem(SDM_MEM64);
1121:         break;
1122:     case MEM_TYPE_128:
1123:         taskResources.SetMem(SDM_MEM128);
1124:         break;
1125:     case MEM_TYPE_256:
1126:         taskResources.SetMem(SDM_MEM256);
1127:         break;
1128:     case MEM_TYPE_512:
1129:         taskResources.SetMem(SDM_MEM512);
1130:         break;
1131:     default:
1132:         break;
1133:     }
1134:     uiCurBufferOffset += sizeof(char);
1135:     msgTask.resources = taskResources.GetUShort();
1136:     //
1137:     // Get the mode
1138:     char cMode = GET_CHAR(&msgCommand.data[uiCurBufferOffset]);
1139:     switch( cMode )
1140:     {
1141:     case EXECUTION_MODE_NORMAL:
1142:         msgTask.mode = MODE_NORMAL;
1143:         break;
1144:     case EXECUTION_MODE_ALWAYS_RUNNING:
1145:         msgTask.mode = MODE_ALWAYS_RUNNING;
1146:         break;
1147:     default:
1148:         break;
1149:     }
1150:     uiCurBufferOffset += sizeof(char);
1151:     //

```

```

1152:    // Get the Preferred Node Id
1153:    char cPnode = GET_CHAR(&msgCommand.data[uiCurBufferOffset]);
1154:    if ( cPnode > 0 ) // Only set a node ID if greater than 0, 0 implies the TM will choose a node
1155:    {
1156:        taskResources.SetPreferredPmNodeId(cPnode);
1157:    }
1158:    uiCurBufferOffset += sizeof(char);
1159:    //
1160:    // Get the task retrieval location
1161:    char cRetrievalLocation = GET_CHAR(&msgCommand.data[uiCurBufferOffset]);
1162:    // Retrieval location is the "version" to use
1163:    msgTask.version = cRetrievalLocation;
1164:
1165:    msgTask.Marshal(buf);
1166:    PostTask(buf);
1167: }
1168: // Command to kill a task
1169: else if (msgCommand.command_id == CMD_KILL_TASK)
1170: {
1171:    // Create a kill request and send
1172:    SDMKill msgKill;
1173:    msgKill.PID = GET_UINT(msgCommand.data);
1174:    msgKill.killLevel = GET_UCHAR(&msgCommand.data[4]);
1175: #ifdef PNP_BACKUP
1176:    if (!isBackup)
1177:    {
1178:        PerformKill(msgKill);
1179:    }
1180: #else
1181:    PerformKill(msgKill);
1182: #endif
1183: }
1184: delete[] buf;
1185: return NULL;
1186: }
1187: /*
1188: SerreqstHandler handles the receipt of an SDMSerreqst message. If the specified message and
interface IDs are appropriate for
1189: one of the TaskManager's service requests messages, the request is handled.
1190: INPUTS:
1191:     arg - The buffer containing the SDMSerreqst message.

```

```

1192: RETURNS:
1193:     void * - Always NULL.
1194: */
1195: void* SerreqstHandler(void* arg)
1196: {
1197:     char* buf=(char*)arg;
1198:     SDMSerreqst msgRequest;
1199:     SDMData msgReply;
1200:
1201:     // Unmarshal the message
1202:     if (msgRequest.Unmarshal(buf) < 0)
1203:     {
1204:         printf("Invalid SDMCommand message. \n");
1205:         return NULL;
1206:     }
1207:     MessageReceived(&msgRequest);
1208:
1209:     debug_f(1,"Service request received (interface %d, message %d)
\n",msgRequest.command_id.getInterface(), msgRequest.command_id.getMessage());
1210:
1211:     // Request to translate a task name to the SDM process identifier
1212:     if (msgRequest.command_id == SER_NAME_TO_PID)
1213:     {
1214:         debug_f(2, " Task name to PID request for task \"%s \". \n", msgRequest.data);
1215:         msgReply.source = msgRequest.source;
1216:         msgReply.msg_id = SER_NAME_TO_PID_REPLY;
1217:
1218:         // Find the pid for the filename given
1219:         pthread_mutex_lock(&list_mutex);
1220:         unsigned int SDMTaskID = taskList.FindPID(msgRequest.data);
1221:         pthread_mutex_unlock(&list_mutex);
1222:
1223:         // Put the filename and the PID into the reply message
1224:         strncpy(msgReply.msg, msgRequest.data, MAX_FILENAME_SIZE);
1225:         PUT_INT(msgReply.msg + MAX_FILENAME_SIZE, SDMTaskID);
1226:
1227:         // Send the message
1228:         const int ReplyLength = msgRequest.length + sizeof(unsigned int);
1229:         msgReply.length = ReplyLength;
1230:         msgReply.Send(msgRequest.destination);
1231:

```

```

1232:     debug_f(1, "    Name to PID reply sent to 0x%lx port: %i.id: %i \n",
msgRequest.destination.getAddress(), msgRequest.destination.getPort(), SDMTaskID);
1233: }
1234: else if (msgRequest.command_id == SER_GET_TASK_LIST)
1235: {
1236:     debug_f(2, " Request to get the task list. \n");
1237:     msgReply.source = msgRequest.source;
1238:     msgReply.msg_id = SER_TASK_LIST_REPLY;
1239:
1240:     unsigned int uiReplyLength;
1241:     pthread_mutex_lock(&list_mutex);
1242:     uiReplyLength = taskList.FillTaskListBlob(msgReply.msg, sizeof(msgReply.msg));
1243:     pthread_mutex_unlock(&list_mutex);
1244:
1245:     msgReply.Send(msgRequest.destination, uiReplyLength);
1246: }
1247: else if (msgRequest.command_id == SER_GET_RUNNING_TASK_LIST)
1248: {
1249:     debug_f(2, " Request to get the running task list. \n");
1250:     msgReply.source = msgRequest.source;
1251:     msgReply.msg_id = SER_RUNNING_TASK_LIST_REPLY;
1252:
1253:     unsigned int uiReplyLength;
1254:     pthread_mutex_lock(&list_mutex);
1255:     uiReplyLength = taskList.FillTaskListBlobRunningOnly(msgReply.msg,
sizeof(msgReply.msg));
1256:     pthread_mutex_unlock(&list_mutex);
1257:
1258:     msgReply.Send(msgRequest.destination, uiReplyLength);
1259: }
1260: else if (msgRequest.command_id == SER_GET_TASK_INFO)
1261: {
1262:     debug_f(2, " Request to get task info. \n");
1263:     msgReply.source = msgRequest.source;
1264:     msgReply.msg_id = SER_TASK_INFO_REPLY;
1265:
1266:     unsigned int ulRequestPid = GET_UINT(msgRequest.data);
1267:
1268:     unsigned int uiReplyLength;
1269:     pthread_mutex_lock(&list_mutex);
1270:     uiReplyLength = taskList.FillTaskInfoRequest(ulRequestPid, msgReply.msg,
sizeof(msgReply.msg));

```

```

1271:    pthread_mutex_unlock(&list_mutex);
1272:
1273:    msgReply.Send(msgRequest.destination, uiReplyLength);
1274: }
1275:
1276: return NULL;
1277: }
1278:
1279:
1280: void* KillHandler(void* arg)
1281: {
1282:     char* buf=(char*)arg;
1283:     SDMKill kill;
1284:
1285:     if (kill.Unmarshal(buf) < 0)
1286:     {
1287:         printf("Invalid SDMKill message. \n");
1288:         return NULL;
1289:     }
1290:     MessageReceived(&kill);
1291:     PerformKill(kill);
1292:
1293:     return NULL;
1294: }
1295:
1296: bool PerformKill(SDMKill& msgKill)
1297: {
1298:     debug_f(1, "Kill request for SDM pid %lu \n",msgKill.PID);
1299:
1300:     pthread_mutex_lock(&list_mutex);
1301:     const SDMComponent_ID PMComponentID = taskList.GetPMAddress(msgKill.PID);
1302:
1303:     if (debug >= 3)
1304:     {
1305:         taskList.PrintList();
1306:     }
1307:     pthread_mutex_unlock(&list_mutex);
1308:
1309:     msgKill.Send(PMComponentID);
1310:     return true;
1311: }

```



```

1312:
1313: /*
1314: ReadyHandler handles the receipt of SDMReady messages from a ProcessManager. A PM node
must verify the existence of the
1315: TaskManager through the ready message. The TaskManager responds with an SDMReady
message containing the PM node's
1316: IP address and the IP address of the DataManager.
1317: INPUTS:
1318:     arg - The HandlerArguments object with the SDMReady message.
1319: RETURNS:
1320:     void * - Always NULL
1321: */
1322: void* ReadyHandler(void* arg)
1323: {
1324:     HandlerArguments *hArgs = static_cast<HandlerArguments*>(arg);
1325:     const char* buf = hArgs->getBuffer();
1326:     unsigned long addr;
1327:     SDMReady msgReady;
1328:     SDMReady msgReadyRequest;
1329:     //bool setNewDMAAddress = false;
1330:
1331:     // Unmarshal the message
1332:     if (msgReadyRequest.Unmarshal(buf) < 0)
1333:     {
1334:         printf("Invalid SDMReady message. \n");
1335:         delete hArgs;
1336:         return NULL;
1337:     }
1338:     MessageReceived(&msgReadyRequest);
1339:
1340:     debug_f(3,"Ready query received \n");
1341:
1342:     // If the SDMReady message is from the DataManager
1343:     if (msgReadyRequest.destination.getPort() == PORT_DM)
1344:     {
1345:         delete hArgs;
1346:         return NULL;
1347:     }
1348:     // If this is from the Task Manager's monitor process
1349:     else if (msgReadyRequest.source.getPort() == PORT_TM_MONITOR)
1350:     {

```

```

1351:    // Send the reply message
1352:    debug_f(3,"Received TM monitor heartbeat message, responding. \n");
1353:    msgReady.Send(msgReadyRequest.source);
1354:    MessageSent(&msgReady);
1355:    delete hArgs;
1356:    return NULL;
1357: }
1358: // Get the IP address of the sender
1359: addr = hArgs->getSenderHost();
1360: msgReady.destination.setPort(PORT_PM);
1361:
1362: // If the PM is on the same node as the TM, and the TM already knows its IP address, give the PM
the known IP
1363: if(addr == inet_addr("127.0.0.1") && Address_TM != inet_addr("127.0.0.1"))
1364:     msgReady.destination.setAddress(Address_TM);
1365: else
1366:     msgReady.destination.setAddress(addr);
1367: // The DM's address, is given to the PM at the source component ID
1368: msgReady.source = DataManager;
1369: msgReady.Send(msgReady.destination);
1370: MessageSent(&msgReady);
1371: /*
1372: if (setNewDMAAddress)
1373:     VerifyDM();
1374: */
1375: delete hArgs;
1376: return NULL;
1377: }
1378: /*
1379: TaskFinishedHandler handles the receipt of SDMTTaskFinished messages from a ProcessManager
node. A PM will send an SDMTTaskFinished
1380: message once a task has finished execution, and the task should be removed from the
TaskManager's task list. The TM will also
1381: published the TaskFinished message as described in the TM's xTEDS.
1382: INPUTS:
1383:     arg - The buffer containing the SDMTTaskFinished message.
1384: RETURNS:
1385:     void * - Always NULL.
1386: */
1387: void* TaskFinishedHandler(void* arg)
1388: {
1389:     char* buf=(char*)arg;

```

```

1390: char taskname[MAX_FILENAME_SIZE];
1391: int exitcode;
1392: SDMTaskFinished msgFinished;
1393:
1394: // Unmarshal the message
1395: if (msgFinished.Unmarshal(buf) < 0)
1396: {
1397:     printf("Invalid SDMTaskFinished message. \n");
1398:     return NULL;
1399: }
1400: MessageReceived(&msgFinished);
1401:
1402: debug_f(1,"Task %s has finished, pid %d \n",msgFinished.filename, msgFinished.pid);
1403:
1404: // Decrement the number of running tasks for this PM
1405: if (!pmList.TaskFinished(msgFinished.source))
1406: {
1407:     debug_f(0, "Error, Could not find the PM corresponding to a finished task. \n");
1408: }
1409:
1410: pthread_mutex_lock(&list_mutex);
1411: if (!taskList.TaskFinished(msgFinished.pid))
1412: {
1413:     debug_f(0, "Error, Could not find the task in the task list. \n");
1414: }
1415: pthread_mutex_unlock(&list_mutex);
1416:
1417: if (debug >= 3)
1418: {
1419:     pthread_mutex_lock(&list_mutex);
1420:     taskList.PrintList();
1421:     pthread_mutex_unlock(&list_mutex);
1422: }
1423:
1424: // Get the filename of the finished task
1425: strncpy(taskname, msgFinished.filename, MAX_FILENAME_SIZE);
1426: taskname[MAX_FILENAME_SIZE - 1] = '\0';
1427: exitcode = GET_INT(&msgFinished.result);
1428: #ifdef PNP_BACKUP
1429: if (!isBackup)
1430: {

```

```

1431:    // Publish the task finished notification message
1432:    PublishTaskFinishedMessage(taskname, exitcode);
1433: }
1434: #else
1435: PublishTaskFinishedMessage(taskname, exitcode, msgFinished.pid);
1436: #endif
1437: return NULL;
1438: }
1439: /*
1440: PMRunningTaskHandler handles SDMTTask messages received from ProcessManager nodes.
These messages are sent after the
1441: TaskManager has experienced a fault, and has died. This function rebuilds the pmList so that
tasks don't have to
1442: be reposted and the scheduler for the TM knows how many tasks are running on all registered
PMs.
1443: INPUTS:
1444:     arg - A pointer to a buffer containing the SDMTTask message.
1445: RETURNS:
1446:     void * - Always NULL.
1447: */
1448: void* PMRunningTaskHandler(void* arg)
1449: {
1450:     char* buf = (char*)arg;
1451:     SDMTTask msgTask;
1452:
1453:     // Unmarshal the message
1454:     if (msgTask.Unmarshal(buf) < 0)
1455:     {
1456:         printf("Invalid SDMTTask message. \n");
1457:         return NULL;
1458:     }
1459:     MessageReceived(&msgTask);
1460:
1461:     debug_f(1,"SDMTTask message received. \n");
1462:     //
1463:     // Make sure the TM pid value stays at a consistent value
1464:     pthread_mutex_lock(&pid_mutex);
1465:     if (msgTask.pid >= NextPID)
1466:         NextPID = msgTask.pid + 1;
1467:     pthread_mutex_unlock(&pid_mutex);
1468:     //
1469:     // Add this task to the task list

```

```

1470: Task task;
1471: task.SetTask(SCHEDULED, msgTask.priority, msgTask.pid, 0u /*resources*/,
msgTask.filename, 0 /*interval*/, MODE_NORMAL, 0 /*version*/);
1472: pthread_mutex_lock(&list_mutex);
1473: taskList.AddTo(task);
1474: debug_f(4, "Posting task. \n");
1475: if (debug >= 4)
1476:     taskList.PrintList();
1477: pthread_mutex_unlock(&list_mutex);
1478:
1479:
1480: // If a PM registered before receiving all of its running task messages,
1481: // simply add this task to that list, if this fails, assume the PM hasn't registered
1482: if (pmList.TaskHasStarted(msgTask.source))
1483: {
1484:     // PM is already registered, finish
1485:     return NULL;
1486: }
1487: // Search the pending PMs list to see if this task message corresponds to an already existing PM
1488: if (pending_pms.AlreadyRegistered(msgTask.source))
1489: {
1490:     pending_pms.TaskHasStarted(msgTask.source);
1491: }
1492: else
1493: {
1494:     // If the PM hasn't been found, it needs to be added to the list
1495:     pending_pms.RegisterPM(msgTask.source, 0);
1496:     pending_pms.TaskHasStarted(msgTask.source);
1497: }
1498:
1499: return NULL;
1500: }
1501: /*
1502: Scheduler schedules queued tasks in the system to ProcessManager nodes whose resource types
match. This scheduler routine tries
1503: to schedule tasks even among PM nodes.
1504: INPUTS:
1505:     arg - Not used, exists to match function signature for thread routines.
1506: RETURNS:
1507:     void * - Always NULL.
1508:

```

```

1509: //TODO: This algorithm needs to be updated to be able to schedule more robustly for heterogenous
PM architectures. Currently,
1510: taskList.FindPendingTask() returns the top pending task in the list, if PM resources for that task
hasn't registered,
1511: it will race the rest of the list from being scheduled.
1512: */
1513: void* Scheduler(void* arg)
1514: {
1515:     bool pending_tasks;
1516:     unsigned int uiCurSeconds = 0;
1517:     unsigned int uiCurUSeconds = 0;
1518:     unsigned int uiLastSeconds = 0;
1519:     const unsigned int PREFERRED_SCHEDULE_TIMEOUT = 60;    // In seconds
1520:     const unsigned int WAIT_FOR_PM_TIMEOUT = 100000; // In useconds
1521:     while(1)
1522:     {
1523:         // search for active (i.e. available tasks)
1524:         pthread_mutex_lock(&list_mutex);
1525:         pending_tasks = taskList.AnyPending();
1526:         pthread_mutex_unlock(&list_mutex);
1527:         if(pending_tasks)
1528:         {
1529:             Task TaskData;
1530:             pm_record PMData;
1531:             bool bPmFound = false;
1532:             //
1533:             // Find a task to schedule
1534:             pthread_mutex_lock(&list_mutex);
1535:             int TaskIndex = taskList.FindPendingTask(TaskData /*output*/ );
1536:             if (TaskIndex == -1)
1537:             {
1538:                 pthread_mutex_unlock(&list_mutex);
1539:                 usleep(WAIT_FOR_PM_TIMEOUT);
1540:                 continue;
1541:             }
1542:             //
1543:             // Make sure at least one PM is registered
1544:             if (pmList.IsEmpty())
1545:             {
1546:                 pthread_mutex_unlock(&list_mutex);
1547:                 usleep(WAIT_FOR_PM_TIMEOUT);

```

```

1548:     continue;
1549: }
1550: //
1551: // Find a PM to schedule this task to
1552: if (TaskData.resources.IsPreferredPmIdSet())
1553: {
1554:     // Find the PM with matching node id
1555:     bPmFound = pmList.FindPMID(TaskData.resources, PMData);
1556:     if (!bPmFound)
1557:     {
1558:         // If the PM is not yet available, check the timeout to make sure this task
1559:         // eventually gets scheduled somewhere even if the PM never does register
1560:         SDM_GetTime(&uiCurSeconds, &uiCurUSeconds);
1561:
1562:         if (uiLastSeconds == 0)
1563:             uiLastSeconds = uiCurSeconds;
1564:         else if(uiCurSeconds - uiLastSeconds > PREFERRED_SCHEDULE_TIMEOUT)
1565:         {
1566:             // Remove the preferred id for all tasks in the task
1567:             // list with this current id, the pm node doesn't look
1568:             // like it's going to register
1569:             debug_f(1, "Preferred PM node not found, removing it from the task list... \n");
1570:             taskList.RemovePreferredPmId(TaskData.resources);
1571:             if (debug >= 3)
1572:                 taskList.PrintList();
1573:         }
1574:         else
1575:         {
1576:             debug_f(2, "Preferred PM not found for the top task, waiting for %ul more seconds... \n",
1577:                 PREFERRED_SCHEDULE_TIMEOUT - (uiCurSeconds - uiLastSeconds));
1578:             if (debug >= 3)
1579:                 taskList.PrintList();
1580:         }
1581:         pthread_mutex_unlock(&list_mutex);
1582:         usleep(WAIT_FOR_PM_TIMEOUT);
1583:         continue;
1584:     }
1585: }
1586: else
1587: {

```

```

1588:    // No preferred pm id --
1589:    // Find some PM node to schedule taking into account
1590:    // a relatively uniform node load (task count)
1591:    for (int j = 0; j < MAX_JOBS; j++)
1592:    {
1593:        bPmFound = pmList.FindEligiblePM(j, TaskData.resources, PMData /*output*/ );
1594:        if (bPmFound)
1595:            break;
1596:    }
1597:    if (!bPmFound)
1598:    {
1599:        debug_f(2, "No PM available to schedule tasks. \n");
1600:        // PM not available, wait for one
1601:        pthread_mutex_unlock(&list_mutex);
1602:        usleep(WAIT_FOR_PM_TIMEOUT);
1603:        continue;
1604:    }
1605: }
1606:
1607: //
1608: // At this point, we have both a task and a pm to schedule
1609: //
1610: // Sleep to allow for task coordination
1611: if (TaskData.timeout != 0)
1612: {
1613:     pthread_mutex_unlock(&list_mutex); // Don't hold this over a sleep
1614:     sleep(TaskData.timeout);
1615:
1616:     // Make sure a reset didn't occur while we were sleeping, if so, start over
1617:     pthread_mutex_lock(&g_mutexSchedulerReset);
1618:     bool bResetScheduler = g_bSchedulerReset;
1619:     if (g_bSchedulerReset)
1620:         g_bSchedulerReset = false; // Reset to false
1621:     pthread_mutex_unlock(&g_mutexSchedulerReset);
1622:
1623:     // If there has been a reset, don't continue
1624:     if (bResetScheduler)
1625:         continue;
1626:
1627:     // Otherwise, continue scheduling this task
1628:     pthread_mutex_lock(&list_mutex);

```



```

1629:     }
1630:     //
1631:     // Assign a PID
1632:     pthread_mutex_lock(&pid_mutex);
1633:     TaskData.pid = taskList[TaskIndex].pid = NextPID++;
1634:     pthread_mutex_unlock(&pid_mutex);
1635:
1636: #ifdef PNP_BACKUP
1637:     if (!isBackup)
1638:     {
1639:         SendTask(PMData, TaskData);
1640:     }
1641: #else
1642:     SendTask(PMData, TaskData);
1643: #endif
1644:     //
1645:     // A match, increment the task count
1646:     pmList.TaskHasStarted(PMData.component_id);
1647:     //
1648:     // Save the PM address for the Task
1649:     taskList[TaskIndex].state = SCHEDULED;
1650:     taskList[TaskIndex].timeout = 0;
1651:     taskList[TaskIndex].pmComponentID = PMData.component_id;
1652:     //
1653:     // Debug output
1654:     debug_f(1, "Scheduling task \" %s \" (pid %d) to Process Manager (NodeID %d) with \n  ",
1655:             TaskData.filename, TaskData.resources.GetPreferredPmId(), TaskData.pid);
1656:     if (debug >= 1)
1657:         printResources(PMData.resources);
1658:     debug_f(1, " \n  %d running tasks \n \n", PMData.tasks+1);
1659:     if (debug >= 3)
1660:         taskList.PrintList();
1661:     pthread_mutex_unlock(&list_mutex);
1662:     if (debug >= 3)
1663:         pmList.PrintList();
1664:
1665:     uiLastSeconds = 0;
1666: }
1667: else
1668: {
1669:     // No tasks to schedule, reset the preferred scheduling timeout

```

```

1670:     uiLastSeconds = 0;
1671:     sleep(1);
1672: }
1673: }
1674:
1675: return NULL;
1676: }
1677: /*
1678:  The SendTask sends an appropriate task to an idle PM
1679:  INPUT:
1680:     pm_index - an index into the pmList structure
1681:  RETURNS:
1682:     bool - True if a task has been found to be scheduled, false otherwise.
1683: */
1684: bool SendTask (const pm_record& PMData, const Task& task)
1685: {
1686:     SDMTask msgTask;
1687:
1688:     // if a task was found marshal the task message
1689:     msgTask.priority = task.priority;
1690:     memset(msgTask.filename, '\0', MAX_FILENAME_SIZE);
1691:     strncpy(msgTask.filename, task.filename, MAX_FILENAME_SIZE);
1692:     msgTask.pid = task.pid;
1693:     msgTask.version = task.taskVersion;
1694:
1695:     // Send the message
1696:     msgTask.Send(PMData.component_id);
1697:     MessageSent(&msgTask);
1698:     debug_f(2, "%s task sent \n", task.filename);
1699:
1700:     PublishTaskStartedMessage(task.filename, task.pid);
1701:     return true;
1702: }
1703:
1704: /*
1705:  This thread runs for the life of the TM instance.  This thread is responsible for sending and
1706:  receiving SDMHeartbeat messages to and
1707:  from all registered PM nodes.  Also, if a heartbeat isn't responded to, the this thread will remove
1708:  it from the registered PM
1709:  list, making the assumption that the PM module has failed.
1710: */

```

```

1709: INPUTS:
1710:     arg - Not used.
1711: RETURNS:
1712:     void * - Always NULL.
1713: */
1714: void* PMHeartbeat (void* arg)
1715: {
1716:     sleep(5);
1717:     int num_sent = 0;                //counter for the number of messages sent
1718:     int num_received = 0;           //counter for the number of messages received
1719:
1720: #ifndef __VXWORKS__ //VxWorks does not support poll
1721:     struct pollfd pm_poll_fd;        //poll struct
1722:     pm_poll_fd.events = POLLIN | POLLPRI;
1723:     pm_poll_fd.fd = pm_heartbeat_pipe[0];
1724: #else
1725:     struct fd_set fds;
1726:     FD_SET(pm_heartbeat_pipe, &fds);
1727:     timeval pmTimeout;
1728:     pmTimeout.tv_sec = PM_HEARTBEAT_TIMEOUT;
1729: #endif
1730:
1731:     int poll_res = 0;
1732:     SDMHeartbeat msgHeartbeat;        //heartbeat message to send out
1733:     SDMHeartbeat msgHeartbeatReceived; //heartbeat message to receive
1734:     char buf[BUFSIZE];                //buffer for receiving message
1735:     long length;                      //Length of the heartbeat message being read
1736:
1737:     msgHeartbeat.source = TaskManager;
1738:
1739:     while (1)
1740:     {
1741: #ifdef PNP_BACKUP
1742:         if (!isBackup)
1743:         {
1744: #endif
1745:             // Send out heartbeat messages to all registered Process Managers
1746:             num_sent = pmList.SendHeartbeatToAll(msgHeartbeat);
1747:
1748:             debug_f(3, "Sent heartbeat to %d PMs. \n", num_sent);
1749:

```

```

1750:    // Wait for responses
1751:    num_received = 0;
1752:
1753:    while (num_sent > num_received)
1754:    {
1755:    #ifndef __VXWORKS__ //VxWorks doesn't support poll, use select instead
1756:        if ((poll_res=poll(&pm_poll_fd,1,PM_HEARTBEAT_TIMEOUT)) < 0)
1757:    #else
1758:        if ((poll_res=select(pm_heartbeat_pipe + 1, &fds, NULL, NULL, &pmTimeout)) < 0)
1759:    #endif
1760:        {
1761:            perror ("PMHeartbeat: Poll error. \n");
1762:            break;
1763:        }
1764:        else if (poll_res > 0)
1765:        {
1766:            // Read message length from the pipe
1767:    #ifndef __VXWORKS__
1768:                read(pm_heartbeat_pipe[0], &length, sizeof(long));
1769:    #else
1770:                read(pm_heartbeat_pipe, &length, sizeof(long));
1771:    #endif
1772:            // Read message from pipe
1773:            if (length > 0 && length < BUFSIZE)
1774:            {
1775:    #ifndef __VXWORKS__
1776:                if (read(pm_heartbeat_pipe[0], buf, length) < 0)
1777:    #else
1778:                if (read(pm_heartbeat_pipe, buf, length) < 0)
1779:    #endif
1780:                {
1781:                    continue;
1782:                }
1783:            }
1784:            num_received++;
1785:            if (msgHeartbeatReceived.Unmarshal(buf) < 0) // buf still gets unmarshaled
1786:            {
1787:                printf("Invalid SDMHeartbeat message. \n");
1788:            }
1789:            else
1790:            {

```

```

1791:         debug_f(3,"PM heartbeat received. \n");
1792:
1793:         // Find the PM node for which this heartbeat corresponds and reset its status
1794:         pmList.HeartbeatReceived(msgHeartbeatReceived.source);
1795:     }
1796: }
1797: // If any PM nodes didn't respond within five seconds, break
1798: else
1799: {
1800:     break;
1801: }
1802: }
1803:
1804: // Check for any PM nodes that haven't responded in a while
1805: pm_record PMNode;
1806: while (pmList.RemoveAnyFailed(PMNode))
1807: {
1808:     debug_f(0,"PM node failed. \n");
1809:     if (debug >= 3)
1810:         pmList.PrintList();
1811:
1812:     pthread_mutex_lock(&list_mutex);
1813:     taskList.PMFailure(PMNode.component_id);
1814:     if (debug >= 3)
1815:         taskList.PrintList();
1816:     pthread_mutex_unlock(&list_mutex);
1817: #ifdef BUILD_FOR_PNPSAT
1818:     // Power-cycle the endpoint
1819:     PowerCyclePmNode(PMNode.component_id);
1820: #endif
1821: }
1822:
1823: // Only send out heartbeats every five seconds or so
1824: sleep(5);
1825: #ifdef PNP_BACKUP
1826: }
1827: #endif
1828: }
1829: return NULL;
1830: }
1831:

```

```

1832: void RegInfoHandler(void* arg)
1833: {
1834:     char* pMessage = static_cast<char*>(arg);
1835:     SDMRegInfo msgRegInfo;
1836:     long lResult = msgRegInfo.Unmarshal(pMessage);
1837:     if (lResult < 0)
1838:     {
1839:         if (lResult != SDM_NO_FURTHER_DATA_PROVIDER)
1840:             printf("Invalid SDMRegInfo message. \n");
1841:         // Don't handle invalid messages or no further data provider messages
1842:         return;
1843:     }
1844:
1845: #ifdef BUILD_FOR_PNPSAT
1846:     if (msgRegInfo.id == REQREG_POWER_CONTROLLER_ID)
1847:     {
1848:         if (msgRegInfo.type == SDM_REGINFO_REGISTRATION)
1849:         {
1850:             pthread_mutex_lock(&g_mutexPowerControllerData);
1851:
1852:             g_mmEndpointReset.setMsgDef(msgRegInfo.msg_def);
1853:             g_midEndpointReset = msgRegInfo.msg_id;
1854:             g_cidPowerController = msgRegInfo.source;
1855:             g_bPowerControllerFound = true;
1856:
1857:             pthread_mutex_unlock(&g_mutexPowerControllerData);
1858:         }
1859:         else if (msgRegInfo.type == SDM_REGINFO_CANCELLATION)
1860:         {
1861:             pthread_mutex_lock(&g_mutexPowerControllerData);
1862:
1863:             g_bPowerControllerFound = false;
1864:
1865:             pthread_mutex_unlock(&g_mutexPowerControllerData);
1866:         }
1867:     }
1868: #endif // BUILD_FOR_PNPSAT
1869: }
1870:
1871:
1872: #ifdef BUILD_FOR_PNPSAT

```

```

1873: void RequestRegPowerController()
1874: {
1875:     SDMReqReg msgRequest;
1876:
1877:     msgRequest.destination = TaskManager;
1878:     msgRequest.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
1879:     msgRequest.id = REQREG_POWER_CONTROLLER_ID;
1880:     strncpy(msgRequest.device, "PowerController", sizeof(msgRequest.device));
1881:     strncpy(msgRequest.interface, "PowerStatusInterface", sizeof(msgRequest.interface));
1882:     strncpy(msgRequest.item_name, "DeviceAsimReset", sizeof(msgRequest.item_name));
1883:
1884:     msgRequest.Send();
1885: }
1886:
1887: /*
1888:  Power cycle a PM node upon failure.
1889:  */
1890: void PowerCyclePmNode(const SDMComponent_ID& idPm)
1891: {
1892:     pthread_mutex_lock(&g_mutexPowerControllerData);
1893:
1894:     if (g_bPowerControllerFound)
1895:     {
1896:         SDMCommand msgCommand;
1897:
1898:         // Set the message id and the source of the command
1899:         msgCommand.command_id = g_mmEndpointReset.getMsgID(COMMANDMSG);
1900:         msgCommand.source = g_cidPowerController;
1901:
1902:         // Set the sensor id of the pm node and the length of the message
1903:         g_mmEndpointReset.setValue("SensorId", msgCommand, idPm.getSensorID());
1904:         msgCommand.length = g_mmEndpointReset.getLength(COMMANDMSG);
1905:
1906:         // Send the command
1907:         msgCommand.Send();
1908:     }
1909:     else
1910:     {
1911:         debug_f(0, "%s -- Error, power controller has not been discovered. \n", __FUNCTION__);
1912:     }
1913:

```

```

1914: pthread_mutex_unlock(&g_mutexPowerControllerData);
1915: }
1916: #endif // BUILD_FOR_PNPSAT
1917:
1918: #ifdef PNP_BACKUP
1919: void ForwardToBackup (char* buf, long length)
1920: {
1921:     list<SDMComponent_ID>::iterator it;
1922:     /* send the buf to each of the backup TM's */
1923:     for (it = Backup_TM_List.begin(); it != Backup_TM_List.end(); ++it)
1924:     {
1925:         in_addr in;
1926:         in.s_addr = it->getAddress();
1927:         char* ip = inet_ntoa(in);
1928:         int port = (int)it->getPort();
1929:         UDPsendto (ip, port, buf, length);
1930:     }
1931: }
1932: #endif
1933:
1934: /*
1935:  Listener is the main message receiving thread for the TaskManager. After a message is received,
this routine spawns off a thread at
1936:  the appropriate handler routine to handle the requests of the message.
1937:  INPUTS:
1938:      arg - Not used, exists to match the function signature for thread routines.
1939:  RETURNS:
1940:      void * - NULL
1941:  */
1942: void* Listener(void* arg)
1943: {
1944:     char buf[BUFSIZE];
1945:     char result;
1946:     bool m_dm_found;
1947:     long length;
1948:     pthread_t HandlerThread;
1949:     pthread_attr_t attr;
1950:
1951: #ifdef PNP_FAKE
1952:     MessageManager* mm = (MessageManager*)arg;
1953: #else

```



```

1954:  MessageManager mm;
1955:  if (!mm.Async_Init_Both(PORT_TM))
1956:  {
1957:      perror("Error initializing message manager \n");
1958:  }
1959: #endif
1960:
1961:  SDMComHandle ComHandle;
1962:
1963:  pthread_attr_init(&attr);
1964:  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
1965:  pthread_attr_setstacksize(&attr, THREAD_STACK_SIZE);
1966:
1967:  while(1)
1968:  {
1969: #ifdef PNP_FAKE
1970:      result = mm->BlockGetMessage(buf /* output */, length /* output */, ComHandle /* output
1971: */);
1972: #else
1973:      result = mm.BlockGetMessage(buf /* output */, length /* output */, ComHandle /* output */);
1974: #endif
1975:      pthread_mutex_lock(&perf_counter_mutex);
1976:      total_recd++;
1977:      prevsec_recd++;
1978:      pthread_mutex_unlock(&perf_counter_mutex);
1979:
1980:      pthread_mutex_lock(&dm_found_mutex);
1981:      m_dm_found = dm_found;
1982:      pthread_mutex_unlock(&dm_found_mutex);
1983:
1984:      if(m_dm_found)
1985:      {
1986:          //      debug_f(1, "=== Received message: %c === \n", buf[0]);
1987:          switch(result)
1988:          {
1989:              case SDM_ACK:
1990:                  ackReceived = true;
1991:                  break;
1992:              case SDM_Register:
1993:                  registerReceived = true;

```

```

1994:         break;
1995:     case SDM_ID:
1996:         idReceived = true;
1997:         break;
1998:     case SDM_RegPM:
1999: #ifdef PNP_BACKUP
2000:         ForwardToBackup(buf, length);
2001: #endif
2002:         RegPMHandler(buf);
2003:         break;
2004:
2005:     case SDM_ReqCode:
2006:         if(pthread_create(&HandlerThread,&attr,&ReqCodeHandler,memcpy(new
char[length],buf,length)) != 0)
2007:         {
2008:             perror("Listener: Could not create ReqCodeHandler thread. \n");
2009:         }
2010:         break;
2011:
2012:     case SDM_PostTask:
2013: #ifdef PNP_BACKUP
2014:         ForwardToBackup(buf, length);
2015: #endif
2016:         PostTaskHandler(buf);
2017:         break;
2018:
2019:     case SDM_RegInfo:
2020: #ifdef PNP_BACKUP
2021:         ForwardToBackup(buf, length);
2022: #endif
2023:         RegInfoHandler(buf);
2024:         break;
2025:
2026:     case SDM_Subreqst:
2027: #ifdef PNP_BACKUP
2028:         ForwardToBackup(buf, length);
2029: #endif
2030:         SubreqstHandler(buf);
2031:         break;
2032:
2033:     case SDM_Deletesub:

```

```

2034: #ifdef PNP_BACKUP
2035:     ForwardToBackup(buf, length);
2036: #endif
2037:     DeletesubHandler(buf);
2038:     break;
2039:
2040:     case SDM_Command:
2041: #ifdef PNP_BACKUP
2042:     ForwardToBackup(buf, length);
2043: #endif
2044:     if(pthread_create(&HandlerThread,&attr,&CommandHandler,memcpy(new
char[length],buf,length)) != 0)
2045:     {
2046:         perror("Listener: Could not create CommandHandler thread. \n");
2047:     }
2048:     break;
2049:
2050:     case SDM_Ready:
2051:         ReadyHandler(new HandlerArguments(buf, length, ComHandle.GetAddress()-
>sin_addr.s_addr, ntohs(ComHandle.GetAddress()->sin_port)));
2052:         break;
2053:
2054:     case SDM_TaskFinished:
2055: #ifdef PNP_BACKUP
2056:         ForwardToBackup(buf, length);
2057: #endif
2058:         TaskFinishedHandler(buf);
2059:         break;
2060:
2061:     case SDM_Task:
2062: #ifdef PNP_BACKUP
2063:         ForwardToBackup(buf, length);
2064: #endif
2065:         PMRunningTaskHandler(buf);
2066:         break;
2067:
2068:     case SDM_Serreqst:
2069:         SerreqstHandler(buf);
2070:         break;
2071:
2072:     case SDM_Kill:

```

```

2073:         KillHandler(buf);
2074:         break;
2075: #ifndef WIN32
2076:         case SDM_Heartbeat:
2077:             QueueHeartbeat(length, buf);
2078:             break;
2079: #endif
2080:
2081:
2082: #ifdef PNP_BACKUP
2083:         case SDM_DMLeader:
2084:             ReceiveLeader(buf);
2085:             break;
2086:
2087:         case SDM_Election:
2088:             exit(EXIT_FAILURE);
2089:             break;
2090: #endif
2091:
2092:         default:
2093:             debug_f(1,"Unexpected message received: %c \n", result);
2094:             break;
2095:     }
2096: }
2097: else
2098:     { // Need to verify the DM
2099:         if (result == SDM_Ready)
2100:         {
2101:             debug_f(3,"Ready message received \n");
2102:             SDMReady in;
2103:             in.Unmarshal(buf);
2104:             if(in.destination.getPort() == PORT_DM)
2105:             {
2106:                 DataManager.setAddress(in.destination.getAddress());
2107:                 debug_f(3,"DM address is 0x%lx:%hd %ld \n",DataManager.getAddress(),
DataManager.getPort(), DataManager.getSensorID());
2108:                 // Address_TM = in.destination.getAddress();
2109:                 pthread_mutex_lock(&dm_found_mutex);
2110:                 dm_found = true;
2111:                 pthread_mutex_unlock(&dm_found_mutex);
2112:             }

```

```

2113:     }
2114: }
2115: }
2116: return NULL;
2117: }
2118:
2119: void QueueHeartbeat (long length, char buf[])
2120: {
2121:     SDMHeartbeat temp;
2122:     temp.Unmarshal(buf);
2123:
2124: #ifdef PNP_BACKUP
2125:     if (Backup_TM_List.end() != find(Backup_TM_List.begin(), Backup_TM_List.end(),
temp.source)) // found in the Backup TM List
2126:     {
2127:         debug_f (3, "Queuing heartbeat for main TM \n");
2128:         /* Write the heartbeat message into the heartbeat pipe for the heartbeat thread */
2129:         // First, write the length of the message
2130:         if (write(tm_main_heartbeat_pipe[1], &length, sizeof(long)) < 0)
2131:             printf("Could not sent heartbeat length to handler thread \n");
2132:         // Write the actual message
2133:         if (write(tm_main_heartbeat_pipe[1], buf, length) < 0)
2134:             printf("Could not send heartbeat to handler thread \n");
2135:     }
2136:     else if (temp.source == mainTM)
2137:     {
2138:         debug_f (3, "Queuing heartbeat for backup TM \n");
2139:         /* Write the heartbeat message into the heartbeat pipe for the heartbeat thread */
2140:         // First, write the length of the message
2141:         if (write(tm_back_heartbeat_pipe[1], &length, sizeof(long)) < 0)
2142:             printf("Could not sent heartbeat length to handler thread \n");
2143:         // Write the actual message
2144:         if (write(tm_back_heartbeat_pipe[1], buf, length) < 0)
2145:             printf("Could not send heartbeat to handler thread \n");
2146:     }
2147:     else
2148:     {
2149: #endif
2150:         debug_f (3, "Queuing heartbeat for PM \n");
2151:         /* Write the heartbeat message into the heartbeat pipe for the heartbeat thread */
2152:         // First, write the length of the message

```

```

2153: #ifndef __VXWORKS__
2154:     if (write(pm_heartbeat_pipe[1], &length, sizeof(long)) < 0)
2155: #else
2156:     if (write(pm_heartbeat_pipe, &length, sizeof(long)) < 0)
2157: #endif
2158:     printf("Could not sent heartbeat length to handler thread \n");
2159:     // Write the actual message
2160: #ifndef __VXWORKS__
2161:     if (write(pm_heartbeat_pipe[1], buf, length) < 0)
2162: #else
2163:     if (write(pm_heartbeat_pipe, buf, length) < 0)
2164: #endif
2165:     printf("Could not send heartbeat to handler thread \n");
2166: #ifdef PNP_BACKUP
2167: }
2168: #endif
2169: }
2170:
2171:
2172: /*
2173: The printResources routine outputs resource requirements in a human readable format
2174: INPUT:
2175:     resources - bitwise or'd constant representing resource requirements
2176: RETURNS:
2177:     void
2178: */
2179: void printResources(const SDMTaskResources& resources)
2180: {
2181:     int os;        //contains the portion of resources pertaining to operating systems
2182:     int mem;       //contains the portion of resources pertaining to memory requirements
2183:     int arch;      //contains the portion of resources pertaining to architectures
2184:     int pmid;      //contains the portion of resources pertaining to process manager id
2185:
2186:     // mask of the various portions of resources
2187:     os = resources.GetOs();
2188:     mem = resources.GetMem();
2189:     arch = resources.GetArch();
2190:     pmid = resources.GetPreferredPmId();
2191:
2192:     debug_f(3, " resources: 0x%x pmid: 0x%x os: 0x%x mem: 0x%x arch: 0x%x\n",
resources.GetUShort(),pmid,os,mem,arch);

```

```

2193:
2194: // print architecture requirements
2195: switch(arch)
2196: {
2197: case SDM_SPAU:
2198:     printf("SPA-U");
2199:     break;
2200: case SDM_INTEL:
2201:     printf("Intel");
2202:     break;
2203: case SDM_PPC_7404:
2204:     printf("Power PC 7404");
2205:     break;
2206: case SDM_PPC_755:
2207:     printf("Power PC 755");
2208:     break;
2209: case SDM_PPC_405:
2210:     printf("Power PC 405");
2211:     break;
2212: case SDM_MICROBLAZE:
2213:     printf("Microblaze");
2214:     break;
2215: case SDM_SPARC:
2216:     printf("Sparc");
2217:     break;
2218: default:
2219:     printf("Unknown architecture");
2220:     break;
2221: }
2222:
2223: printf (" machine, running ");
2224:
2225: // print os requirements
2226: switch(os)
2227: {
2228: case SDM_LINUX24:
2229:     printf("Linux 2.4");
2230:     break;
2231: case SDM_LINUX26:
2232:     printf("Linux 2.6");
2233:     break;

```

```

2234: case SDM_WIN32:
2235:     printf("Win32");
2236:     break;
2237: break;
2238: case SDM_VXWORKS:
2239:     printf("VxWorks");
2240:     break;
2241: default:
2242:     printf("Unknown OS");
2243:     break;
2244: }
2245: printf(" with at least ");
2246: // print memory requirements
2247: switch(mem)
2248: {
2249: case SDM_MEM32:
2250:     printf("32 Mb RAM");
2251:     break;
2252: case SDM_MEM64:
2253:     printf("64 Mb RAM");
2254:     break;
2255: case SDM_MEM128:
2256:     printf("128 Mb RAM");
2257:     break;
2258: case SDM_MEM256:
2259:     printf("256 Mb RAM");
2260:     break;
2261: case SDM_MEM512:
2262:     printf("512 Mb RAM");
2263:     break;
2264: case SDM_MEM1024:
2265:     printf("1024 Mb RAM");
2266:     break;
2267: default:
2268:     printf("Unknown memory");
2269:     break;
2270: }
2271: printf(" \n");
2272: }
2273:
2274: /*

```



```

2275: The SendCode routine sends executable code to a PM
2276: INPUT:
2277:     filename - the name of the file to be sent
2278: RETURNS:
2279:     void
2280: */
2281: void SendCode (char* buf)    // sends no file on failure
2282: { int fd;                    //file descriptor to the code to be sent
2283:   int length;                //length of code packet to send
2284:   SDLError err;              //error message
2285:   char file[MAX_FILENAME_SIZE];
2286:   int max_code_size;
2287:   unsigned short sequence = 0;
2288:   unsigned short n_messages = 0;
2289:   unsigned short seq_to_send = 0;
2290:   SDMCode msgCode;
2291:   SDMReqCode msgRequest;
2292:   struct stat file_stats;
2293:   if (msgRequest.Unmarshal(buf) < 0)
2294:   {
2295:       printf("Invalid SDMReqCode message. \n");
2296:       return;
2297:   }
2298:   MessageReceived(&msgRequest);
2299:
2300:   debug_f(2,"PM has requested code for %s. \n",msgRequest.filename);
2301:   memset(file,0,MAX_FILENAME_SIZE);
2302:   memcpy(file,msgRequest.filename,strlen(msgRequest.filename)+1);
2303:   max_code_size = sizeof(msgCode.code);
2304:   //
2305:   // Determine which file version is to be sent
2306:   char strFullFilePath[256];
2307:   switch (msgRequest.version)
2308:   {
2309:   case TASK_LOCATION_PRIMARY:
2310:       snprintf(strFullFilePath, sizeof(strFullFilePath), "%s/%s",
2311:           STR_TASK_LOCATION_PRIMARY, msgRequest.filename);
2312:       debug_f(2, " Using primary location %s to open file. \n", strFullFilePath);
2313:       break;
2314:   case TASK_LOCATION_TEMPORARY:
2315:       snprintf(strFullFilePath, sizeof(strFullFilePath), "%s/%s",

```

```

2316:         STR_TASK_LOCATION_TEMPORARY, msgRequest.filename);
2317:     debug_f(2, " Using temporary location %s to open file. \n", strFullFilePath);
2318:     break;
2319: case TASK_LOCATION_BACKUP:
2320:     snprintf(strFullFilePath, sizeof(strFullFilePath), "%s/%s",
2321:         STR_TASK_LOCATION_BACKUP, msgRequest.filename);
2322:     debug_f(2, " Using backup location %s to open file. \n", strFullFilePath);
2323:     break;
2324: default:
2325:     strcpy(strFullFilePath, msgRequest.filename);
2326:     debug_f(2, " Using default location to open file. \n");
2327: }
2328:
2329: #ifdef __VXWORKS__
2330:     strcat(strFullFilePath, ".vxe");
2331: #endif
2332:
2333: // open filename for reading
2334: #ifdef WIN32
2335:     if ((fd = open(strFullFilePath, O_RDONLY | _O_BINARY)) < 0)
2336: #else
2337:     if ((fd = open(strFullFilePath, O_RDONLY)) < 0)
2338: #endif
2339:     {
2340:         printf("Error opening file: %s for code transfer \n", strFullFilePath);
2341:         err.error = SDM_ERROR_CODE_NOT_FOUND;
2342:         strncpy(err.error_msg, msgRequest.filename, sizeof(err.error_msg));
2343:         err.SendTo(msgRequest.source);
2344:         MessageSent(&err);
2345:     }
2346: else
2347:     {
2348: #ifdef WIN32
2349:         pthread_mutex_lock(&code_transfer_mutex);
2350: #endif
2351:         /*Get file size*/
2352: #ifdef __VXWORKS__
2353:         if(stat(strFullFilePath, &file_stats) < 0)
2354:         {
2355:             printf("Error statting file: %s \n",strerror(errno));
2356:         }

```

```

2357:     int temp = (int)file_stats.st_size;
2358:     n_messages = (unsigned short)(temp/max_code_size);
2359:     if (((temp/((double)max_code_size)) > ((double)n_messages))
2360:     {
2361:         n_messages++;
2362:     }
2363: #else
2364:     if(fstat(fd, &file_stats) < 0)
2365:     {
2366:         printf("fstat() return an error: %s \n", strerror(errno));
2367:     }
2368:     n_messages = (unsigned short)(file_stats.st_size/max_code_size);
2369:     if (((file_stats.st_size/((double)max_code_size)) > ((double)n_messages))
2370:         n_messages++;
2371: #endif
2372:     msgCode.num_segments = n_messages;
2373:     /*If the PM is rerequesting a specified sequence number*/
2374:     if (msgRequest.seq_num > 0)
2375:     {
2376:         debug_f(2, " (Rerequest for packet %d) \n", msgRequest.seq_num);
2377:         seq_to_send = msgRequest.seq_num;
2378:         /*Seek to the sequence number*/
2379:         lseek(fd, (seq_to_send-1)*max_code_size, SEEK_SET);
2380:         length = read(fd, &msgCode.code, max_code_size);
2381:
2382:         if (length > 0)
2383:             msgCode.code_length = length;
2384:         msgCode.seq_num = seq_to_send;
2385:         msgCode.num_segments = n_messages;
2386:         strcpy(msgCode.filename, file);
2387:         msgCode.SendTo(msgRequest.source);
2388:         MessageSent(&msgCode);
2389:         close(fd);
2390:     }
2391:     /*If this is a code request for the entire file*/
2392:     else
2393:     {
2394:         lseek(fd, 0, SEEK_SET);
2395:         length = read (fd, &msgCode.code, max_code_size);
2396:         if(length > 0)
2397:         {

```

```

2398:         msgCode.code_length = length;
2399:     }
2400:     while (length > 0)
2401:     {
2402:         // send code packet
2403:         msgCode.seq_num = ++sequence;
2404:         msgCode.num_segments = n_messages;
2405:         strcpy(msgCode.filename, file);
2406: #ifdef WIN32
2407:         Sleep(500);
2408: #endif
2409:         msgCode.SendTo(msgRequest.source);
2410:         MessageSent(&msgCode);
2411:
2412:         debug_f(2,"Sending code message for %s of code length %d with checksum %x.
2413: \n",file,length,msgCode.c_sum);
2414:         // read next packet
2415:         length = read (fd, &msgCode.code, max_code_size);
2416:         if(length > 0)
2417:         {
2418:             msgCode.code_length = length;
2419:         }
2420:     }
2421:     debug_f(2,"file sent. \n");
2422:     // close file
2423:     close(fd);
2424: }
2425: #ifdef WIN32
2426:     pthread_mutex_unlock(&code_transfer_mutex);
2427: #endif
2428: }
2429: }
2430:
2431: /*
2432:  The PostTask routine queues a task for execution
2433:  INPUT:
2434:      buf - the SDMPostTask message
2435:  RETURNS:
2436:      void
2437:  */

```

```

2438: void PostTask (char *buf)
2439: {
2440:     Task task;
2441:     SDMPostTask msgPostTask;
2442:     // unmarshal message
2443:     if (msgPostTask.Unmarshal(buf) < 0)
2444:     {
2445:         printf("Invalid SDMPostTask message. \n");
2446:         return;
2447:     }
2448:     MessageReceived(&msgPostTask);
2449:
2450:     // Fill the task info
2451:     task.SetTask(PENDING, msgPostTask.priority, 0, msgPostTask.resources,
2452:         msgPostTask.filename, msgPostTask.sched_interval,
2453:         msgPostTask.mode, msgPostTask.version);
2454:
2455:     pthread_mutex_lock(&list_mutex);
2456:     taskList.AddTo(task);
2457:     debug_f(4, "Posting task. \n");
2458:     if (debug >= 4)
2459:     {
2460:         taskList.PrintList();
2461:     }
2462:     pthread_mutex_unlock(&list_mutex);
2463:
2464:     if (debug >= 2)
2465:     {
2466:         debug_f(2, "%s task posted requiring a ", task.filename);
2467:         printResources(task.resources);
2468:         debug_f(2, ". \n");
2469:     }
2470: #ifdef PNP_BACKUP
2471:     if (!isBackup)
2472:     {
2473:         PublishTaskQueuedMessage(task.filename);
2474:     }
2475: #else
2476:     PublishTaskQueuedMessage(task.filename);
2477: #endif
2478: }

```

```

2479: /*
2480:  VerifyDM verifies the existence of the DataManager. The DM is necessary for the TaskManager
to carry out its needed operations.
2481:  The DataManager's response to the SDMReady message is received in the Listener routine.
2482:  INPUTS:
2483:      None.
2484:  RETURNS:
2485:      void
2486: */
2487: void VerifyDM(void)
2488: {
2489:  bool msg_rcvd = false;
2490:  SDMReady msgReadyOut;
2491:  msgReadyOut.destination.setPort(PORT_TM);
2492:  msgReadyOut.destination.setAddress(TaskManager.getAddress());
2493:  //msgReadyOut.destination.setAddress(DataManager.getAddress());
2494:  printf("Searching for DM .");
2495:  do
2496:  {
2497:      pthread_mutex_lock(&perf_counter_mutex);
2498:      total_sent++;
2499:      prevsec_sent++;
2500:      pthread_mutex_unlock(&perf_counter_mutex);
2501:      // Ready messages sent via broadcast to find the DM
2502:      // TODO: need to test broadcast under uclinux!
2503:      //msgReadyOut.SendBCast(DataManager.getAddress(), PORT_DM);
2504:      msgReadyOut.Send(DataManager);
2505:      putchar('.');
2506:      //fflush(NULL);
2507:      usleep(500000);
2508:      pthread_mutex_lock(&dm_found_mutex);
2509:      msg_rcvd = dm_found;
2510:      pthread_mutex_unlock(&dm_found_mutex);
2511:  } while(!msg_rcvd);
2512:  printf("DM found \n");
2513: }
2514:
2515: /*
2516:  InitialStartUp parses through the sdm.config file found in the current directory to startup any
tasks specified in that file.
2517:  INPUTS:

```

```

2518:     None.
2519: RETURNS:
2520:     void
2521: */
2522: void InitialStartUp(void)
2523: {
2524:     SDMPostTask msgTask;
2525:     char buf[256];
2526:     int fd = 0;
2527:     int value;
2528:     unsigned int i = 0;
2529: #ifndef __VXWORKS__
2530:     char file[65536];
2531: #else
2532:     char file[32768];
2533: #endif
2534:     int cont = 0;
2535:     char filename[MAX_FILENAME_SIZE];
2536:     int name = 0;
2537:     unsigned int resource = 0;
2538:     int start = 0;
2539:     int end = 0;
2540:     int tempi = 0;
2541:     int timeout = 0;
2542:     int mode = 0;
2543:     int mem = 0;
2544:     char memstring[8];
2545:
2546:     // Open the SDM.config file
2547:     fd = open(FILE_SDM_TASK_LIST, O_RDONLY);
2548:     if(fd == -1)
2549:     {
2550:         perror("SDM task list was unable to be opened! ");
2551:         printf("Could not open  \"%%s \" \"\n", FILE_SDM_TASK_LIST);
2552:         return;
2553:     }
2554: #ifndef __VXWORKS__
2555:     value = read(fd,file,65535);
2556: #else
2557:     value = read(fd,file,32767);
2558: #endif

```

```

2559: close(fd);
2560: if(value < 0)
2561: {
2562:     printf("Could read from file  \"%s \"\n", FILE_SDM_TASK_LIST);
2563:     return;
2564: }
2565: file[value] = '\0';
2566:
2567: bool TagClosed = false, inComment = false;;
2568: const unsigned int FILE_LENGTH = (int)strlen(file);
2569: for(i = 0; i < FILE_LENGTH; i++)
2570: {
2571:     if(strncmp(&file[i], "<!--", 4) == 0)
2572:     {
2573:         i+= 4;
2574:         inComment = true;
2575:         char* cp = strstr(&file[i], "-->");
2576:         if ( cp == NULL )
2577:         {
2578:             printf("%s doesn't close the comment started at char pos %d\n",
2579:                 FILE_SDM_TASK_LIST,i);
2580:             return;
2581:         }
2582:         i = (int)(cp-file)+3;
2583:     }
2584:     //if(strncmp(&file[i], "Device", 6) == 0)
2585:     //{
2586:     //    // Don't do anything with devices
2587:     //}
2588:     if(strncmp(&file[i], "Application", 11) == 0)
2589:     {
2590:         cont = 1;
2591:         name = 0;
2592:         resource = 0;
2593:         timeout = 0;
2594:         mode = 0;
2595:         memset(filename, 0, sizeof(filename));
2596:         i+=11;
2597:     }
2598:     if(strncmp(&file[i], "pathOnSpacecraft", 16) == 0)
2599:     {

```



```

2600:     name = 1;
2601:     i+=16+1;
2602:     while(file[i] != "")
2603:     {
2604:         if(file[i] == '\0')
2605:             return;
2606:         i++;
2607:     }
2608:     i++;
2609:     // Get the name of the executable
2610:     start = i;
2611:     while(file[i] != "")
2612:     {
2613:         i++;
2614:     }
2615:     end = i;
2616:     if(end - start > MAX_FILENAME_SIZE)
2617:         memcpy(filename,&file[start],MAX_FILENAME_SIZE);
2618:     else
2619:         memcpy(filename,&file[start],(end-start));
2620:     debug_f(3,"Start of filename is %d and end is %d with filename %s\n",start,end,filename);
2621: }
2622: if(strncmp(&file[i],"architecture",12) == 0)
2623: {
2624:     i+=12+1;
2625:     while(file[i] != "")
2626:     {
2627:         if(file[i] == '\0')
2628:             return;
2629:         i++;
2630:     }
2631:     i++;
2632:     if(strncmp(&file[i],"SPAU",4) == 0)
2633:         resource |= SDM_SPAU;
2634:     else if(strncmp(&file[i],"Intel",5) == 0)
2635:         resource |= SDM_INTEL;
2636:     else if(strncmp(&file[i],"PPC7404",7) == 0)
2637:         resource |= SDM_PPC_7404;
2638:     else if(strncmp(&file[i],"PPC755",6) == 0)
2639:         resource |= SDM_PPC_755;

```

```

2640:         else if(strncmp(&file[i],"PPC405",6) == 0)
2641:             resource |= SDM_PPC_405;
2642:         else if(strncmp(&file[i],"Microblaze",10) == 0)
2643:             resource |= SDM_MICROBLAZE;
2644:         else if(strncmp(&file[i],"Sparc",5) == 0)
2645:             resource |= SDM_SPARC;
2646:         cont |= 0x02;
2647:     }
2648: if(strncmp(&file[i],"memoryMinimum",13) == 0)
2649: {
2650:     i+=13+1;
2651:     while(file[i] != "")
2652:     {
2653:         if(file[i] == '\0')
2654:             return;
2655:         i++;
2656:     }
2657:     i++;
2658:     tempi = i;
2659:     while(file[i] != "")
2660:     {
2661:         if(file[i] == '\0')
2662:             return;
2663:         i++;
2664:     }
2665:     memset(memstring,0,8);
2666:     memcpy(memstring,&file[tempi],i-tempi);
2667:     mem = atoi(memstring);
2668:     if(mem <= 32)
2669:         resource |= SDM_MEM32;
2670:     else if(mem <= 64)
2671:         resource |= SDM_MEM64;
2672:     else if(mem <= 128)
2673:         resource |= SDM_MEM128;
2674:     else if(mem <= 256)
2675:         resource |= SDM_MEM256;
2676:     else if(mem <= 512)
2677:         resource |= SDM_MEM512;
2678:     else if(mem <= 1024)
2679:         resource |= SDM_MEM1024;
2680:     cont |= 0x04;

```

```

2681:     }
2682:     if(strncmp(&file[i], "operatingSystem", 15) == 0)
2683:     {
2684:         i+=15+1;
2685:         while(file[i] != "")
2686:         {
2687:             if(file[i] == '\0')
2688:                 return;
2689:             i++;
2690:         }
2691:         i++;
2692:         if(strncmp(&file[i], "Win32", 5) == 0)
2693:             resource |= SDM_WIN32;
2694:         else if(strncmp(&file[i], "Linux24", 7) == 0)
2695:             resource |= SDM_LINUX24;
2696:         else if(strncmp(&file[i], "Linux26", 7) == 0)
2697:             resource |= SDM_LINUX26;
2698:         else if(strncmp(&file[i], "VxWorks", 7) == 0)
2699:             resource |= SDM_VXWORKS;
2700:         cont |= 0x08;
2701:     }
2702:     if (cont == 15 && name == 1)
2703:     {
2704:         if (file[i] == '>')
2705:             TagClosed = true;
2706:     }
2707:     if (strncmp(&file[i], "preferredPMNodeID", 17) == 0)
2708:     {
2709:         i+=17+1;
2710:         while (file[i] != "")
2711:         {
2712:             if (file[i] == '\0')
2713:                 return;
2714:             i++;
2715:         }
2716:         i++;
2717:         char strNodeId[8];
2718:         int Offset = 0;
2719:         while (file[i + Offset] != "")
2720:         {
2721:             strNodeId[Offset] = file[i + Offset];

```

```

2722:         Offset++;
2723:
2724:         // This should not happen
2725:         if (Offset == sizeof(strNodeId) - 1)
2726:             break;
2727:     }
2728:     strNodeId[Offset] = '\0';
2729:
2730:     int ID = atoi(strNodeId);
2731:     if (ID < 1 || ID > 255)
2732:     {
2733:         debug_f(0, "Invalid preferredPMNodeID value in %s. \n",
FILE_SDM_TASK_LIST);
2734:         continue;
2735:     }
2736:     resource |= PM_ID(atoi(strNodeId));
2737: }
2738: if (strncmp(&file[i], "scheduleTimeout", 15) == 0)
2739: {
2740:     i+=15+1;
2741:     while (file[i] != "")
2742:     {
2743:         if (file[i] == '\0')
2744:             return;
2745:         i++;
2746:     }
2747:     i++;
2748:     char strTimeout[8];
2749:     int Offset = 0;
2750:     while (file[i + Offset] != "")
2751:     {
2752:         strTimeout[Offset] = file[i + Offset];
2753:         Offset++;
2754:
2755:         // This should not happen
2756:         if (Offset == sizeof(strTimeout) - 1)
2757:             break;
2758:     }
2759:     strTimeout[Offset] = '\0';
2760:     timeout = atoi(strTimeout);
2761: }

```

```

2762:     if (strncmp(&file[i], "executionMode", 13) == 0)
2763:     {
2764:         i+=13+1;
2765:         while (file[i] != "")
2766:         {
2767:             if (file[i] == '\0')
2768:             {
2769:                 return;
2770:             }
2771:             i++;
2772:         }
2773:
2774:         i++;
2775:         if (strncmp(&file[i], "alwaysRunning", 13) == 0)
2776:         {
2777:             mode = MODE_ALWAYS_RUNNING;
2778:         }
2779:     }
2780:     if(cont == 15 && name == 1 && TagClosed == true)
2781:     {
2782:         msgTask.resources = resource;
2783:         msgTask.sched_interval = timeout;
2784:         msgTask.mode = mode;
2785:         strncpy(msgTask.filename, filename, MAX_FILENAME_SIZE);
2786:         msgTask.Marshal(buf);
2787:         PostTask(buf);
2788:         cont = 0;
2789:         name = 0;
2790:         TagClosed = false;
2791:     }
2792: }
2793: return;
2794: }
2795:
2796: /*
2797:  MessageRecieved logs any messages received from the TaskManager (if logging is enabled).
  This function should be called any
2798:  time an SDM* message is received within the TaskManager.
2799:  INPUTS:
2800:      msg - The message to be logged.
2801:  RETURNS:

```

```

2802:     void
2803: */
2804: void MessageReceived(SDMmessage *msg)
2805: {
2806:     pthread_mutex_lock(&log_service_mutex);
2807:     if (!log_service.IsEmpty())
2808:         log_service.MessageReceived(msg);
2809:     pthread_mutex_unlock(&log_service_mutex);
2810: }
2811:
2812: /*
2813:  MessageSend logs any message sent from the TaskManager (if logging is enabled). This routine
also increments the TaskManager's
2814:  messaging performance counters. This function should be called any time an SDM* message is
sent within the TaskManager.
2815:  INPUTS:
2816:      msg - The message to be logged.
2817:  RETURNS:
2818:      void
2819: */
2820: void MessageSent(SDMmessage *msg)
2821: {
2822:     /*Increment performance counters*/
2823:     pthread_mutex_lock(&perf_counter_mutex);
2824:     total_sent++;
2825:     prevsec_sent++;
2826:     pthread_mutex_unlock(&perf_counter_mutex);
2827:     /*Log the message, if the logger is enabled*/
2828:     pthread_mutex_lock(&log_service_mutex);
2829:     if (!log_service.IsEmpty())
2830:         log_service.MessageSent(msg);
2831:     pthread_mutex_unlock(&log_service_mutex);
2832: }
2833:
2834: // -----
2835: // -----FUNCTIONS PERTAINING TO BACKUP-----
2836: // -----
2837:
2838: #ifdef PNP_BACKUP
2839:
2840: #ifdef PNP_FAKE

```

```

2841: /*
2842:  * sendIMA sends a message to the NetworkManager. This should only be called if the
TaskManager is a backup and the main has gone down.
2843: */
2844: void sendIMA (void)
2845: {
2846:     cout << "Sending IMA \n";
2847:
2848:     // send message to NM
2849:     SDMDMLeader ima;
2850:     ima.source = TaskManager;
2851:     ima.SendTo(NetworkManager);
2852:
2853:     sleep (5); // wait for election to finish
2854: }
2855: #endif
2856:
2857: #ifdef BUILD_FOR_PNPSAT
2858: void RequestDMInfo (void)
2859: {
2860:     struct hostent *he;
2861:     unsigned long addr;
2862:
2863:     while ((he=gethostbyname("datamanager.spacewire")) == NULL)
2864:     {
2865:         sleep(1);
2866:     }
2867:     memcpy(&addr, he->h_addr, sizeof(addr));
2868:
2869:     in_addr in;
2870:     in.s_addr = addr;
2871:
2872:     DataManager.setAddress(addr);
2873:     DataManager.setPort(PORT_DM);
2874: }
2875:
2876: bool AmIBackup (void)
2877: {
2878:     cout << "Determining leader/backup status \n";
2879:
2880:     SendIMA(SdmImaTm, debug);

```

```

2881:  sleep(5); //Wait 5 seconds while NM resolves election
2882:
2883:  struct hostent *he;
2884:  unsigned long addr;
2885:
2886:  he=gethostbyname("taskmanager.spacewire");
2887:  memcpy(&addr, he->h_addr, sizeof(addr));
2888:
2889:  if (addr == TaskManager.getAddress())
2890:  {
2891:      return false;
2892:  }
2893:
2894:  mainTM.setAddress(addr);
2895:  mainTM.setPort(PORT_TM);
2896:
2897:  // alert Main TM that I am a backup
2898:
2899:  return true;
2900: }
2901:
2902: #endif
2903:
2904: #ifdef PNP_FAKE
2905: void RequestDMInfo (MessageManager* mm)
2906: {
2907:     cout << "Requesting DM info from NM \n";
2908:     char buf[BUFSIZE];
2909:     char flag = '\0';
2910:     // Request DM info from NM
2911:     SDMDMLLeader req;
2912:     req.source.setAddress(Address_TM);
2913:     req.source.setPort(PORT_TM);
2914:     req.running_flag = 'd';
2915:
2916:     req.SendTo(NetworkManager);
2917:
2918:     while (flag != 'd')
2919:     {
2920:         mm->BlockGetMessage(buf);
2921:

```



```

2922:   SDMDMLLeader temp;
2923:   temp.Unmarshal(buf);
2924:   flag = temp.running_flag;
2925:
2926:   ReceiveLeader(buf);
2927: }
2928: }
2929:
2930: void AmIBackup (MessageManager* mm)
2931: {
2932:   cout << "Determining leader/backup status \n";
2933:   char buf[BUFSIZE];
2934:
2935:   SDMDMLLeader ima;
2936:
2937:   ima.source = TaskManager;
2938:   ima.SendTo(NetworkManager);
2939:
2940:   mm->BlockGetMessage(buf);
2941:
2942:   ReceiveLeader(buf);
2943: }
2944: #endif
2945:
2946: void TakeOverLeadPosition (void)
2947: {
2948:   // alert DM
2949:   SDMReady msg;
2950:   msg.destination = TaskManager;
2951:   msg.SendTo(DataManager);
2952:
2953:   // alert all PM's
2954:   SDMDMLLeader pmMsg;
2955:   pmMsg.source = TaskManager;
2956:   pmMsg.running_flag = 't';
2957:
2958:   pmList.SendToAll(pmMsg);
2959:
2960:   pmList.PrintList();
2961:   taskList.PrintList();
2962: }

```

```

2963:
2964: void VerifyNewDM(char *sender_buf)
2965: {
2966:     SDMComponent_ID tempDM;
2967:     tempDM.Unmarshal(sender_buf, 0);
2968:     tempDM.setPort(3504);
2969:     bool msg_rcvd = false;
2970:     SDMReady msgReadyOut;
2971:     msgReadyOut.destination.setPort(PORT_TM);
2972:     msgReadyOut.destination.setAddress(TaskManager.getAddress());
2973:     printf("Searching for DM at: 0x%lx port: %u.", tempDM.getAddress(), tempDM.getPort());
2974:     fflush(NULL);
2975:     do
2976:     {
2977:         pthread_mutex_lock(&perf_counter_mutex);
2978:         total_sent++;
2979:         prevsec_sent++;
2980:         pthread_mutex_unlock(&perf_counter_mutex);
2981:         msgReadyOut.SendTo(tempDM);
2982:         putchar('.');
2983:         fflush(NULL);
2984:         usleep(500000);
2985:         pthread_mutex_lock(&dm_found_mutex);
2986:         msg_rcvd = dm_found;
2987:         pthread_mutex_unlock(&dm_found_mutex);
2988:     } while(!msg_rcvd);
2989:     printf("DM found \n");
2990:     DataManager.setAddress(tempDM.getAddress());
2991: }
2992:
2993: void ReceiveLeader (char* buf)
2994: {
2995:     SDMDMLeader temp;
2996:     temp.Unmarshal(buf);
2997:
2998:     debug_f (1, "Running flag is %c \n", temp.running_flag);
2999:
3000:     cout << "Running flag == " << temp.running_flag << endl;
3001:
3002:     switch (temp.running_flag)
3003:     {

```

```

3004:   case '0':
3005:       // I'm the leader now
3006:       isBackup = false;
3007:       debug_f(1, "I'm now the main TM \n");
3008:       TakeOverLeadPosition();
3009:       break;
3010:
3011:   case '1':
3012:       debug_f(1, "I've been assigned as backup \n");
3013:       debug_f(1, "Alerting leader to my presence \n");
3014:       mainTM = temp.source; // using destination to pass the mainTM's comp id
3015:       temp.source = TaskManager;
3016:       temp.running_flag = '2';
3017:       temp.SendTo (mainTM);
3018:       break;
3019:
3020:   case '2':
3021:       debug_f(1, "Adding 0x%x as backup \n", temp.source.getAddress());
3022:       Backup_TM_List.push_back(temp.source);
3023:       break;
3024:
3025:   case 'd':
3026:       debug_f(1, "Received %lu from NM as DM's address \n", temp.source.getAddress());
3027:       cout << "Received " << temp.source << " from NM as DM's address" << endl;
3028:       DataManager.setAddress(temp.source.getAddress());
3029:       DataManager.setPort(PORT_DM);
3030:       break;
3031:
3032:   default:
3033:       break;
3034: }
3035: }
3036:
3037: /*
3038:  This thread runs for the life of the TM instance.  This thread is responsible for sending and
3039:  receiving SDMHeartbeat messages to and
3040:  from all registered PM nodes.  Also, if a heartbeat isn't responded to, the this thread will remove
3041:  it from the registered PM
3042:  list, making the assumption that the PM module has failed.
3043:
3044:  This thread runs differently if you are a backup TM.  Backup's shouldn't be heartbeating the PM's,
3045:  instead they should heartbeat the main

```

```

3043:  TM.
3044:
3045:  INPUTS:
3046:      arg - Not used.
3047:  RETURNS:
3048:      void * - Always NULL.
3049: */
3050:
3051: void* TMHeartbeat (void* arg)
3052: {
3053:     sleep(5);
3054:
3055:     struct pollfd tm_back_poll_fd;           //poll struct
3056:     tm_back_poll_fd.events = POLLIN | POLLPRI;
3057:     tm_back_poll_fd.fd = tm_back_heartbeat_pipe[0];
3058:
3059:     struct pollfd tm_main_poll_fd;           //poll struct
3060:     tm_main_poll_fd.events = POLLIN | POLLPRI;
3061:     tm_main_poll_fd.fd = tm_main_heartbeat_pipe[0];
3062:
3063:     int poll_res = 0;
3064:     SDMHeartbeat msgHeartbeat;               //heartbeat message to send out
3065:     SDMHeartbeat msgHeartbeatReceived;       //heartbeat message to receive
3066:     char buf[BUFSIZE];                       //buffer for receiving message
3067:     long length;                             //Length of the heartbeat message being read
3068:
3069:     msgHeartbeat.source = TaskManager;
3070:
3071:     while (1)
3072:     {
3073:         if (isBackup)
3074:         {
3075:             bool mainIsAlive = true;
3076:             SDMHeartbeat pulse;
3077:             pulse.source = TaskManager;
3078:
3079:             while (mainIsAlive)
3080:             {
3081:                 cout << "Sending heartbeat to leader TM at " << mainTM << endl;
3082:                 cout.flush();
3083:                 // send heart beat

```

```

3084:    pulse.SendTo(mainTM);
3085:
3086:    // poll for response, with a 5 second timeout
3087:    poll_res = poll (&tm_back_poll_fd,1,5000);
3088:
3089:    if (poll_res < 0)
3090:    {
3091:        perror ("Heartbeat: Poll error. \n");
3092:    }
3093:    else if (poll_res == 0) // when poll returns a zero it means there was a time out
3094:    {
3095:        printf ("Main task manager not responding, sending IMA message \n");
3096:        mainIsAlive = false;
3097:    }
3098:    else // clear the pipe by reading the message
3099:    {
3100:        // Read message length from the pipe
3101:        read(tm_back_heartbeat_pipe[0], &length, sizeof(long));
3102:
3103:        // Read message from pipe
3104:        if (length > 0 && length < BUFSIZE)
3105:        {
3106:            read(tm_back_heartbeat_pipe[0], buf, length);
3107:
3108:            SDMHeartbeat temp;
3109:            temp.Unmarshal(buf);
3110:        }
3111:    }
3112:
3113:    if (mainIsAlive)
3114:    {
3115:        // only send heart beats every 5 seconds
3116:        sleep(5);
3117:    }
3118: }
3119:
3120:
3121: #ifdef PNP_FAKE
3122:     sendIMA();
3123: #endif
3124:

```

```

3125: #ifdef SEND_IMA
3126:     if (spacewire)
3127:     {
3128:         SendIMA(ImaTm, debug);
3129:     }
3130: #endif
3131: }
3132: else
3133: {
3134:     while (1)
3135:     {
3136:         //respond to any heart beats we have received while sleeping
3137:         if ((poll_res=poll(&tm_main_poll_fd,1,1000)) < 0)
3138:         {
3139:             perror ("PMHeartbeat: Poll error. \n");
3140:             break;
3141:         }
3142:         else if (poll_res > 0)
3143:         {
3144:             // Read message length from the pipe
3145:             read(tm_main_heartbeat_pipe[0], &length, sizeof(long));
3146:
3147:             // Read message from pipe
3148:             if (length > 0 && length < BUFSIZE)
3149:             {
3150:                 read(tm_main_heartbeat_pipe[0], buf, length);
3151:
3152:                 // send a reply
3153:
3154:                 SDMHeartbeat temp;
3155:                 temp.Unmarshal (buf);
3156:
3157:                 SDMComponent_ID backup;
3158:                 backup = temp.source;
3159:
3160:                 temp.source = TaskManager;
3161:                 temp.SendTo(backup);
3162:                 cout << "Replied to heartbeat from " << backup << endl;
3163:             }
3164:         }
3165:     }

```

```
3166: }  
3167: }  
3168: return NULL;  
3169: }  
3170:  
3171: #endif // ifdef PNP_BACKUP
```

File: sdm/tm/tasklist.cpp

```
1: // TaskList implementation file
2:
3: #include "tasklist.h"
4: #include "../common/Exception/SDMBadIndexException.h"
5: #include "../common/message/SDMTaskError.h"
6: #include <string.h>
7: #include <unistd.h>
8: #include <stdio.h>
9:
10: TaskList::TaskList()
11: {   Init();
12: }
13:
14: void TaskList::Init()
15: {
16:     for (int n=0; n<NUMTASKS; n++)
17:     {   tasks[n].InitTask();
18:     }
19: }
20:
21: bool TaskList::AnyInactive()
22: {
23:     for (int n=0; n<NUMTASKS; n++)
24:         if (tasks[n].state == INACTIVE)
25:             return true;
26:     return false;
27: }
28:
29: bool TaskList::AnyPending()
30: {
31:     for (int n=0; n<NUMTASKS; n++)
32:         if (tasks[n].state == PENDING)
33:             return true;
34:     return false;
35: }
36:
37: bool TaskList::AddTo(const Task& t)
38: {
39:     int n;
```



```

40: if (!AnyInactive()) return false;
41: else
42: {
43:     for (n=0; n<NUMTASKS && tasks[n].state != INACTIVE; n++) ;
44:     if (n >= NUMTASKS) return false;
45:     else tasks[n] = t;
46: }
47: return true;
48: }
49:
50: bool TaskList::RemoveFrom(unsigned int p) // remove task by pid
51: {
52: int n;
53: for (n=0; n<NUMTASKS && tasks[n].pid != p; n++) ;
54: if (n >= NUMTASKS) return false;
55: else tasks[n].Delete();
56: return true;
57: }
58:
59: bool TaskList::RemoveFrom(char *filename)
60: {
61: int n;
62: if (filename == NULL)
63:     return false;
64: for (n=0; n<NUMTASKS; n++)
65: {
66:     if (tasks[n].filename != NULL)
67:         if (!strcmp(tasks[n].filename, filename))
68:             break;
69: }
70: if (n >= NUMTASKS)
71:     return false;
72: else
73:     tasks[n].Delete();
74: return true;
75: }
76:
77: /*
78: Perform whatever cleanup is needed when a PM reports that a task is finished.
79: Take the following actions depending on the task's mode:
80:  MODE_NORMAL -- The task is finished, remove it from the list.

```

```

81:  MODE_ALWAYS_RUNNING -- The task probably faulted, reschedule it.
82: */
83: bool TaskList::TaskFinished(unsigned int a_uiPid)
84: {
85:  int n = 0;
86:  bool bFound = false;
87:  bool bResult = false;
88:
89:  // Find the finished task
90:  for (n = 0; n < NUMTASKS; n++)
91:  {
92:    if (tasks[n].pid == a_uiPid)
93:    {
94:      bFound = true;
95:      break;
96:    }
97:  }
98:
99:  // If the task wasn't found, return error
100:  if (false == bFound)
101:  {
102:    return false;
103:  }
104:
105:  // Otherwise, n is the index of the task
106:  switch(tasks[n].taskMode)
107:  {
108:    case MODE_NORMAL:
109:      // Remove the task from the list
110:      bResult = RemoveFrom(a_uiPid);
111:      break;
112:    case MODE_ALWAYS_RUNNING:
113:      // Set the task to be rescheduled, remove its pid
114:      tasks[n].state = PENDING;
115:      tasks[n].pid = 0;
116:      bResult = true;
117:      break;
118:    default:
119:      printf("%s -- Invalid mode option. \n", __FUNCTION__);
120:      break;
121:  }

```

```

122:
123: return bResult;
124: }
125:
126: void TaskList::ClearList()
127: {
128:     for (int n = 0; n < NUMTASKS; n++)
129:     {
130:         tasks[n].Delete();
131:     }
132: }
133:
134: bool TaskList::IsFilePresent(char *f)
135: {
136:     int n;
137:     for (n=0; n<NUMTASKS; n++)
138:         if (tasks[n].filename != NULL)
139:             if (!strcmp(f, tasks[n].filename))
140:                 return true;
141:     return false;
142: }
143:
144: unsigned int TaskList::SetState (unsigned int p, char state)           // set state by pid
145: {
146:     int n;
147:     for (n=0; n<NUMTASKS; n++)
148:         if (tasks[n].pid == p)
149:         {   tasks[n].state = state;
150:             if (state == FINISHED)
151:                 tasks[n].pid = PID_INVALID;
152:             return tasks[n].pid;
153:         }
154:     return 0;
155: }
156:
157: /*bool MatchResources (int pmResources, int taskResources)           // verify   pmResources   <=
taskResources
158: {
159:     if ((taskResources & PMIDMASK) != 0)
160:         if ((pmResources & PMIDMASK) != (taskResources & PMIDMASK))
161:             return false;

```

```

162:
163:   if ((pmResources & ARCHMASK) == (taskResources & ARCHMASK))
164:       if ((pmResources & MEMMASK) >= (taskResources & MEMMASK))
165:           if ((pmResources & OSMASK) == (taskResources & OSMASK))
166:               return true;
167:   return false;
168: }*/
169:
170: // Find a pending task in the list
171: int TaskList::FindPendingTask(Task& taskOut)
172: {
173:   for (int n = 0; n < NUMTASKS; n++)
174:   {
175:       if (tasks[n].state == PENDING)
176:       {
177:           taskOut = tasks[n];
178:           return n;
179:       }
180:   }
181:   return -1;
182: }
183:
184: // Only match PENDING tasks (tasks that have not been scheduled)
185: bool TaskList::FindPendingMatch (int resources, unsigned int newPID, Task& taskOut)
186: {
187:   int n, lastpos;
188:   bool success = false;
189:   taskOut.InitTask();
190:   lastpos=0;
191:   for (n=0; n<NUMTASKS && !success; n++)
192:   {   if (tasks[n].state == PENDING)
193:       if (SDMTaskResources::MatchResources (resources, tasks[n].resources))
194:           if (tasks[n].priority >= taskOut.priority)
195:           {
196:               tasks[n].pid = newPID;
197:               taskOut = tasks[n];
198:               success = true;
199:               lastpos = n;
200:           }
201:   }
202:   return success;

```

```

203: }
204:
205: bool TaskList::SetAddress(unsigned int p, const SDMComponent_ID& PM)
206: {
207:     int n;
208:     bool found = false;
209:     for (n=0; n<NUMTASKS && !found; n++)
210:     {
211:         if (tasks[n].pid == p)
212:         {
213:             tasks[n].SetPM(PM);
214:             return true;
215:         }
216:     }
217:     return false;
218: }
219:
220: void TaskList::RemovePreferredPmId(const SDMTaskResources& TaskResources)
221: {
222:     for (int n = 0; n < NUMTASKS; n++)
223:     {
224:         if (tasks[n].state != INACTIVE && tasks[n].resources.GetPreferredPmId() ==
TaskResources.GetPreferredPmId())
225:             tasks[n].resources.SetPreferredPmNodeId(0);
226:     }
227: }
228:
229: unsigned int TaskList::FindPID(const char* filename)
230: {
231:     if (filename == NULL)
232:         return PID_INVALID;
233:     for (int n = 0; n < NUMTASKS; n++)
234:     {
235:         if (tasks[n].state == SCHEDULED && tasks[n].filename != NULL &&
strcmp(tasks[n].filename, filename)==0)
236:         {
237:             return tasks[n].pid;
238:         }
239:     }
240:     return PID_INVALID;
241: }

```

```

242:
243: // This PM has failed, set all tasks to PENDING and cancel their xTEDS
244: void TaskList::PMFailure(const SDMComponent_ID& PMID)
245: {
246:     SDMTaskError msgError;
247:     msgError.source = PMID;
248:     for (int n = 0; n < NUMTASKS; n++)
249:     {
250:         if (tasks[n].GetPM() == PMID)
251:         {
252:             msgError.source.setSensorID(tasks[n].pid);
253:             msgError.pid = tasks[n].pid;
254:             msgError.Send();
255:
256:             tasks[n].state = PENDING;
257:         }
258:     }
259: }
260:
261: void TaskList::PrintList()
262: {
263:     printf(" \n***Task List is: \n");
264:     for (int n = 0; n < NUMTASKS; n++)
265:     {
266:         if (tasks[n].state != INACTIVE)
267:         {
268:             char StateStr[16];
269:             switch(tasks[n].state)
270:             {
271:                 case INACTIVE:
272:                     strncpy(StateStr, "INACTIVE", sizeof(StateStr)); break;
273:                 case ACTIVE:
274:                     strncpy(StateStr, "ACTIVE", sizeof(StateStr)); break;
275:                 case ASSIGNED:
276:                     strncpy(StateStr, "ASSIGNED", sizeof(StateStr)); break;
277:                 case SCHEDULED:
278:                     strncpy(StateStr, "SCHEDULED", sizeof(StateStr)); break;
279:                 case PENDING:
280:                     strncpy(StateStr, "PENDING", sizeof(StateStr)); break;
281:                 case FINISHED:
282:                     strncpy(StateStr, "FINISHED", sizeof(StateStr)); break;

```

```

283:         }
284:         printf(" Index %d: name: %s pid: %d state: %s priority: %hhhd resources: %hd timeout:
%d mode: %d version: %d \n",n,tasks[n].filename, tasks[n].pid, StateStr, tasks[n].priority,
tasks[n].resources.GetUShort(), tasks[n].timeout, tasks[n].taskMode, tasks[n].taskVersion);
285:     }
286: }
287: printf ("**** \n \n");
288: }
289:
290: SDMComponent_ID TaskList::GetPMAddress(unsigned int pid)
291: {
292:     for (int n = 0; n < NUMTASKS; n++)
293:     {
294:         if (tasks[n].pid == pid)
295:         {
296:             return tasks[n].GetPM();
297:         }
298:     }
299:     return SDMComponent_ID();
300: }
301:
302: Task& TaskList::operator[] (int index)
303: {
304:     if (index < 0 || index >= NUMTASKS)
305:         throw SDMBadIndexException(__FUNCTION__);
306:
307:     if (tasks[index].state == INACTIVE)
308:         throw SDMBadIndexException(__FUNCTION__);
309:
310:     return tasks[index];
311: }
312:
313: unsigned int TaskList::FillTaskListBlob(char* pBuffer, unsigned int uiBufferSize)
314: {
315:     return FillTaskListBlob(pBuffer, uiBufferSize, TASKS_ALL);
316: }
317:
318: unsigned int TaskList::FillTaskListBlobRunningOnly (char* pBuffer, unsigned int uiBufferSize)
319: {
320:     return FillTaskListBlob(pBuffer, uiBufferSize, TASKS_RUNNING);
321: }

```

```

322:
323: /*
324:  * Fill a "blob" buffer of the tasks specified in eTaskSet. The blob consists of
325:  * each task's SDM pid and their task name stored sequentially in pBuffer. Each
326:  * entry is null terminated by the task name, and the last entry is double
327:  * null terminated.
328:  */
329: unsigned int TaskList::FillTaskListBlob(char* pBuffer, unsigned int uiBufferSize, TaskSet
eTaskSet)
330: {
331:     unsigned int uiCurOffset = sizeof(unsigned short);
332:     unsigned int uiBytesRemaining = uiBufferSize - 2; // Include double null term
333:     unsigned short usNumTasks = 0;
334:     char strFormatString[512];
335:
336:     for (int n = 0; n < NUMTASKS; n++)
337:     {
338:         if (tasks[n].state == INACTIVE)
339:             continue;
340:
341:         switch(eTaskSet)
342:         {
343:             case TASKS_RUNNING:
344:                 // TODO: "Scheduled" really means "running" this should be changed
345:                 if (SCHEDULED != tasks[n].state)
346:                     break;
347:
348:                 // else fall through
349:             case TASKS_ALL:
350:                 {
351:                     snprintf(strFormatString, sizeof(strFormatString), "%-8u%s  \n", tasks[n].pid,
tasks[n].filename);
352:                     unsigned int uiCurEntryLength = strlen(strFormatString);
353:
354:                     if (uiCurOffset + uiCurEntryLength < uiBytesRemaining)
355:                     {
356:                         strcpy(&pBuffer[uiCurOffset], strFormatString);
357:
358:                         uiCurOffset += uiCurEntryLength;
359:                         uiBytesRemaining -= uiCurEntryLength;
360:                         usNumTasks++;

```



```

361:     }
362: }
363: break;
364: default:
365:     printf("%s -- Invalid TaskSet. \n", __FUNCTION__);
366:     return 0;
367: }
368: }
369: PUT_USHORT(&pBuffer[0], usNumTasks);
370:
371: pBuffer[uiCurOffset++] = '\0';
372:
373: return uiCurOffset;
374: }
375:
376: unsigned int TaskList::FillTaskInfoRequest(unsigned int uiPid, char* pBuffer, unsigned int
uiBufferSize)
377: {
378: // Find the task
379: int iTaskIndex = 0;
380: for ( ; iTaskIndex < NUMTASKS; iTaskIndex++)
381: {
382:     if (tasks[iTaskIndex].state == INACTIVE)
383:     {
384:         continue;
385:     }
386:     else if (tasks[iTaskIndex].pid == uiPid)
387:     {
388:         // Found
389:         break;
390:     }
391: }
392: if (iTaskIndex >= NUMTASKS)
393: {
394:     // Not found
395:     return 0;
396: }
397:
398: //
399: // Put the task name
400: unsigned int uiCurBufferOffset = 0;

```

```

401: strcpy(pBuffer + uiCurBufferOffset, tasks[iTaskIndex].filename);
402: uiCurBufferOffset += XTEDS_MAX_TASK_NAME_SIZE;
403: //
404: // Put the architecture type, in xTEDS enum form
405: switch( tasks[iTaskIndex].resources.GetArch() )
406: {
407:     case SDM_INTEL:
408:         PUT_CHAR( &pBuffer[uiCurBufferOffset], ARCH_TYPE_INTEL );
409:         break;
410:     case SDM_MICROBLAZE:
411:         PUT_CHAR( &pBuffer[uiCurBufferOffset], ARCH_TYPE_MICROBLAZE );
412:         break;
413:     default:
414:         PUT_CHAR( &pBuffer[uiCurBufferOffset], 0 );
415: }
416: uiCurBufferOffset += sizeof(char);
417: //
418: // Put the OS type
419: switch( tasks[iTaskIndex].resources.GetOs() )
420: {
421:     case SDM_LINUX26:
422:         PUT_CHAR( &pBuffer[uiCurBufferOffset], OS_TYPE_LINUX26 );
423:         break;
424:     case SDM_WIN32:
425:         PUT_CHAR( &pBuffer[uiCurBufferOffset], OS_TYPE_WIN32 );
426:         break;
427:     default:
428:         PUT_CHAR( &pBuffer[uiCurBufferOffset], 0 );
429: }
430: uiCurBufferOffset += sizeof(char);
431: //
432: // Put the memory type
433: switch( tasks[iTaskIndex].resources.GetMem() )
434: {
435:     case SDM_MEM64:
436:         PUT_CHAR( &pBuffer[uiCurBufferOffset], MEM_TYPE_64 );
437:         break;
438:     case SDM_MEM128:
439:         PUT_CHAR( &pBuffer[uiCurBufferOffset], MEM_TYPE_128 );
440:         break;
441:     case SDM_MEM256:

```

```

442:   PUT_CHAR( &pBuffer[uiCurBufferOffset], MEM_TYPE_256 );
443:   break;
444: case SDM_MEM512:
445:   PUT_CHAR( &pBuffer[uiCurBufferOffset], MEM_TYPE_512 );
446:   break;
447: default:
448:   PUT_CHAR( &pBuffer[uiCurBufferOffset], 0 );
449: }
450: uiCurBufferOffset += sizeof(char);
451: //
452: // Put the PM id
453:                                     PUT_UCHAR(                                     &pBuffer[uiCurBufferOffset],
tasks[iTaskIndex].resources.GetPreferredPmIdNum() );
454: uiCurBufferOffset += sizeof(char);
455: //
456: // Put the task state
457: switch( tasks[iTaskIndex].state )
458: {
459: case SCHEDULED:
460:   PUT_CHAR( &pBuffer[uiCurBufferOffset], TASK_STATE_SCHEDULED );
461:   break;
462: case PENDING:
463:   // Queued
464:   PUT_CHAR( &pBuffer[uiCurBufferOffset], TASK_STATE_QUEUED );
465:   break;
466: case FINISHED:
467:   // Stopped
468:   PUT_CHAR( &pBuffer[uiCurBufferOffset], TASK_STATE_STOPPED );
469:   break;
470: case RUNNING:
471:   PUT_CHAR( &pBuffer[uiCurBufferOffset], TASK_STATE_RUNNING );
472:   break;
473: default:
474:   PUT_CHAR( &pBuffer[uiCurBufferOffset], 0 );
475: }
476: uiCurBufferOffset += sizeof(char);
477: //
478: // Put the task mode
479: switch( tasks[iTaskIndex].taskMode )
480: {
481: case MODE_NORMAL:

```

```
482:    PUT_CHAR( &pBuffer[uiCurBufferOffset], EXECUTION_MODE_NORMAL );
483:    break;
484: case MODE_ALWAYS_RUNNING:
485:    PUT_CHAR( &pBuffer[uiCurBufferOffset], EXECUTION_MODE_ALWAYS_RUNNING );
486:    break;
487: default:
488:    PUT_CHAR( &pBuffer[uiCurBufferOffset], 0 );
489: }
490: uiCurBufferOffset += sizeof(char);
491:
492: return uiCurBufferOffset;
493: }
```

File: sdm/tm/SdmTaskList.config

```
1: <?xml version="1.0" encoding="utf-8"?>
2:         <SDMConfig xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3:     <xTEDS>
4:         <Application componentKey="SDM_DM" pathForAssembly="C: \IOC \Program Files \SDM
   \producer"
5:             pathOnSpacecraft="producer"          architecture="Intel"          memoryMinimum="128"
   operatingSystem="Linux26"/>
6:     </xTEDS>
7:     <xTEDS>
8:         <Application componentKey="SDM_DM" pathForAssembly="C: \IOC \Program Files \SDM
   \consumer"
9:             pathOnSpacecraft="consumer"          architecture="Intel"          memoryMinimum="128"
   operatingSystem="Linux26"/>
10:    </xTEDS>
11: </xTEDS>
12:     <Application componentKey="SDM_DM" pathForAssembly="C: \IOC \Program Files \SDM
   \converter"
13:         pathOnSpacecraft="converter"          architecture="Intel"          memoryMinimum="128"
   operatingSystem="Linux26"/>
14: </xTEDS>
15: </SDMConfig>
16:
```

File: sdm/tm/HandlerArguments.h

```
1: #ifndef _HANDLER_ARGUMENTS_H_
2: #define _HANDLER_ARGUMENTS_H_
3:
4: // A simple class for passing "multiple" arguments into threads.
5: class HandlerArguments {
6: public:
7:     HandlerArguments(void *buf, size_t bufLen, unsigned long senderHost = 0, unsigned long
senderPort = 0) :
8:         _buf(new char[bufLen]), _bufLen(bufLen), _senderHost(senderHost), _senderPort(senderPort)
9:     {
10:         if (_buf != NULL)
11:             memcpy(_buf, buf, bufLen);
12:     }
13:
14:     ~HandlerArguments()
15:     {
16:         if (_buf != NULL)
17:             delete[] _buf;
18:     }
19:     // Declared, but not defined, fixes warning
20:     HandlerArguments(const HandlerArguments& right);
21:     HandlerArguments& operator=(const HandlerArguments& right);
22:
23:     const char* getBuffer() const { return _buf; }
24:     size_t getBufferSize() const { return _bufLen; }
25:     unsigned long getSenderHost() const { return _senderHost; }
26:     unsigned long getSenderPort() const { return _senderPort; }
27:
28: private:
29:     char* _buf;
30:     size_t _bufLen;
31:     unsigned long _senderHost;
32:     unsigned long _senderPort;
33: };
34:
35: #endif
```

File: sdm/tm/task.cpp

```
1: // Task classes implementation
2:
3: #include "task.h"
4: #include <stdlib.h>
5: #include <stdio.h>
6: #include <string.h>
7:
8: Task::Task():pmComponentID(),state('
                                \0'),priority('
                                \0'),pid(PID_INVALID),resources(0),filename(NULL),timeout(0),taskMode(MODE_NORMAL),taskVer
                                sion(0)
9: {
10: InitTask();
11: }
12:
13: Task::Task(const Task& a):pmComponentID(a.pmComponentID), state(a.state), priority(a.priority),
                                pid(a.pid),
                                resources(a.resources),
                                filename(strdup(a.filename)),timeout(a.timeout),taskMode(a.taskMode),taskVersion(a.taskVersion)
14: {
15: }
16:
17: Task::~~Task()
18: {
19: if(filename != NULL)
20:     free(filename);
21: }
22:
23: void Task::InitTask()
24: {
25: resources = 0;
26: pid = PID_INVALID;
27: filename = NULL;
28: state = INACTIVE;
29: priority = 0;
30: pmComponentID.setAddress(0);
31: pmComponentID.setSensorID(0);
32: pmComponentID.setPort(0);
33: timeout = 0;
34: taskMode = MODE_NORMAL;
35: taskVersion = 0;
36: }
```

```

37:
38: void Task::Delete()
39: {
40: if (filename != NULL)
41:     free(filename);
42: InitTask();
43: }
44:
45: void Task::SetTask (char s, char pr, unsigned int p, const SDMTaskResources& r, char *f, int
atimeout, int aMode, int aVersion)
46: {
47: resources = r;
48: free (filename);
49: filename = (char *) malloc(strlen(f)+1);
50: strcpy (filename, f);
51: state = s;
52: pid = p;
53: priority = pr;
54: timeout = atimeout;
55: taskMode = aMode;
56: taskVersion = aVersion;
57: }
58:
59: Task& Task::operator= (const Task& t)
60: {
61: Delete();
62: SetTask (t.state, t.priority, t.pid, t.resources, t.filename, t.timeout, t.taskMode, t.taskVersion);
63: return *this;
64: }
65:
66: void Task::SetPM(const SDMComponent_ID& PM)
67: {
68: pmComponentID = PM;
69: }

```


File: sdm/tm/task.h

```
1: // Task classes definitions      Cannon 1/05
2: #ifndef __TASK_H_
3: #define __TASK_H_
4:
5: #include <string.h>
6: #include "../common/message/SDMComponent_ID.h"
7: #include "../common/task/SDMTaskResources.h"
8: #include "../common/task/taskdefs.h"
9:
10:
11: class Task
12: {
13: public:
14: Task();
15: Task(const Task&);
16: ~Task();
17: void InitTask ();
18: void Delete ();
19: void SetTask (char state, char priority, unsigned int pid, const SDMTaskResources& resources, char
*filename, int timeout, int aMode, int aVersion);
20: Task& operator= (const Task& t);
21:
22: void SetPM(const SDMComponent_ID& PM);
23: SDMComponent_ID GetPM() const { return pmComponentID; }
24: public:
25: SDMComponent_ID pmComponentID;
26: char state;
27: char priority;
28: unsigned int pid;
29: SDMTaskResources resources;
30: char *filename;
31: int timeout;          // The timeout time needed to wait until the task should be scheduled
32: int taskMode;
33: int taskVersion;
34: };
35:
36: #endif
```

Listing from directory: sdm/VxWorks

File: sdm/VxWorks/includes/endian.h

```
1: #ifndef __endian_h_
2: #define __endian_h_
3:
4: #ifdef _WIN32
5: #define __BYTE_ORDER __LITTLE_ENDIAN
6: #define __LITTLE_ENDIAN
7: #endif
8:
9: //This assumes that the VxWorks host being used is big endian (PPC750FX, SPARC LEON3)
10: #ifdef __VXWORKS__
11: #define __BIG_ENDIAN
12: #endif
13:
14: #endif // __endian_h_
15:
```

File: sdm/VxWorks/includes/byteswap.h

```
1: #ifndef __byteswap_h_
2: #define __byteswap_h_
3:
4: #define bswap_32(x) ( ((0x000000FF&x) << 24) | ((0x0000FF00&x) << 8) | ((0x00FF0000&x) >> 8) |
((0xFF000000&x) >> 24) )
5: #define bswap_16(x) (((x) >> 8) & 0xff) | (((x) & 0xff) << 8))
6: #define bswap_64(x) (((x) & 0xff00000000000000ull) >> 56) \
7: | (((x) & 0x00ff000000000000ull) >> 40) \
8: | (((x) & 0x0000ff0000000000ull) >> 24) \
9: | (((x) & 0x000000ff00000000ull) >> 8) \
10: | (((x) & 0x00000000ff000000ull) << 8) \
11: | (((x) & 0x0000000000ff0000ull) << 24) \
12: | (((x) & 0x000000000000ff00ull) << 40) \
13: | (((x) & 0x00000000000000ffull) << 56))
14:
15:
16: #endif
17:
```

File: sdm/VxWorks/libRegex/pcre_fullinfo.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:         Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_fullinfo(), which returns
42: information about a compiled pattern. */
43:
44:
45: #ifdef HAVE_CONFIG_H
46: #include "config.h"
47: #endif
48:
49: #include "pcre_internal.h"
50:
51:
52: /*****
53: *      Return info about compiled pattern      *
54: *****/
55:
56: /* This is a newer "info" function which has an extensible interface so
57: that additional items can be added compatibly.
58:
59: Arguments:
60: argument_re    points to compiled code
61: extra_data     points extra data, or NULL
62: what           what information is required
63: where          where to put the information
64:
65: Returns:       0 if data returned, negative on error
66: */
67:
68: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
69: pcre_fullinfo(const pcre *argument_re, const pcre_extra *extra_data, int what,
70: void *where)
71: {
72: real_pcre internal_re;
73: pcre_study_data internal_study;
74: const real_pcre *re = (const real_pcre *)argument_re;
75: const pcre_study_data *study = NULL;
76:

```

```

77: if (re == NULL || where == NULL) return PCRE_ERROR_NULL;
78:
79: if (extra_data != NULL && (extra_data->flags & PCRE_EXTRA_STUDY_DATA) != 0)
80:   study = (const pcre_study_data *)extra_data->study_data;
81:
82: if (re->magic_number != MAGIC_NUMBER)
83: {
84:   re = _pcre_try_flipped(re, &internal_re, study, &internal_study);
85:   if (re == NULL) return PCRE_ERROR_BADMAGIC;
86:   if (study != NULL) study = &internal_study;
87: }
88:
89: switch (what)
90: {
91:   case PCRE_INFO_OPTIONS:
92:     *((unsigned long int *)where) = re->options & PUBLIC_OPTIONS;
93:     break;
94:
95:   case PCRE_INFO_SIZE:
96:     *((size_t *)where) = re->size;
97:     break;
98:
99:   case PCRE_INFO_STUDYSIZE:
100:    *((size_t *)where) = (study == NULL)? 0 : study->size;
101:    break;
102:
103:   case PCRE_INFO_CAPTURECOUNT:
104:     *((int *)where) = re->top_bracket;
105:     break;
106:
107:   case PCRE_INFO_BACKREFMAX:
108:     *((int *)where) = re->top_backref;
109:     break;
110:
111:   case PCRE_INFO_FIRSTBYTE:
112:     *((int *)where) =
113:       ((re->flags & PCRE_FIRSTSET) != 0)? re->first_byte :
114:       ((re->flags & PCRE_STARTLINE) != 0)? -1 : -2;
115:     break;
116:
117:   /* Make sure we pass back the pointer to the bit vector in the external

```

```

118: block, not the internal copy (with flipped integer fields). */
119:
120: case PCRE_INFO_FIRSTTABLE:
121: *((const uschar **)where) =
122:   (study != NULL && (study->options & PCRE_STUDY_MAPPED) != 0)?
123:     ((const pcre_study_data *)extra_data->study_data)->start_bits : NULL;
124: break;
125:
126: case PCRE_INFO_LASTLITERAL:
127: *((int *)where) =
128:   ((re->flags & PCRE_REQCHSET) != 0)? re->req_byte : -1;
129: break;
130:
131: case PCRE_INFO_NAMEENTRYSIZE:
132: *((int *)where) = re->name_entry_size;
133: break;
134:
135: case PCRE_INFO_NAMECOUNT:
136: *((int *)where) = re->name_count;
137: break;
138:
139: case PCRE_INFO_NAMETABLE:
140: *((const uschar **)where) = (const uschar *)re + re->name_table_offset;
141: break;
142:
143: case PCRE_INFO_DEFAULT_TABLES:
144: *((const uschar **)where) = (const uschar *)(_pcre_default_tables);
145: break;
146:
147: case PCRE_INFO_OKPARTIAL:
148: *((int *)where) = (re->flags & PCRE_NOPARTIAL) == 0;
149: break;
150:
151: case PCRE_INFO_JCHANGED:
152: *((int *)where) = (re->flags & PCRE_JCHANGED) != 0;
153: break;
154:
155: case PCRE_INFO_HASCORLRF:
156: *((int *)where) = (re->flags & PCRE_HASCORLRF) != 0;
157: break;
158:

```

```
159: default: return PCRE_ERROR_BADOPTION;
160: }
161:
162: return 0;
163: }
164:
165: /* End of pcre_fullinfo.c */
```


File: sdm/VxWorks/libRegex/pcre_chartables.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3: *****/
4:
5: /* This file contains character tables that are used when no external tables
6:  are passed to PCRE by the application that calls it. The tables are used only
7:  for characters whose code values are less than 256.
8:
9:  This is a default version of the tables that assumes ASCII encoding. A program
10:  called dftables (which is distributed with PCRE) can be used to build
11:  alternative versions of this file. This is necessary if you are running in an
12:  EBCDIC environment, or if you want to default to a different encoding, for
13:  example ISO-8859-1. When dftables is run, it creates these tables in the
14:  current locale. If PCRE is configured with --enable-rebuild-chartables, this
15:  happens automatically.
16:
17:  The following #includes are present because without the gcc 4.x may remove the
18:  array definition from the final binary if PCRE is built into a static library
19:  and dead code stripping is activated. This leads to link errors. Pulling in the
20:  header ensures that the array gets flagged as "someone outside this compilation
21:  unit might reference this" and so it will always be supplied to the linker. */
22:
23: #ifdef HAVE_CONFIG_H
24: #include "config.h"
25: #endif
26:
27: #include "pcre_internal.h"
28:
29: const unsigned char _pcre_default_tables[] = {
30:
31: /* This table is a lower casing table. */
32:
33:  0, 1, 2, 3, 4, 5, 6, 7,
34:  8, 9, 10, 11, 12, 13, 14, 15,
35:  16, 17, 18, 19, 20, 21, 22, 23,
36:  24, 25, 26, 27, 28, 29, 30, 31,
37:  32, 33, 34, 35, 36, 37, 38, 39,
38:  40, 41, 42, 43, 44, 45, 46, 47,
39:  48, 49, 50, 51, 52, 53, 54, 55,
```

40: 56, 57, 58, 59, 60, 61, 62, 63,
 41: 64, 97, 98, 99, 100, 101, 102, 103,
 42: 104, 105, 106, 107, 108, 109, 110, 111,
 43: 112, 113, 114, 115, 116, 117, 118, 119,
 44: 120, 121, 122, 91, 92, 93, 94, 95,
 45: 96, 97, 98, 99, 100, 101, 102, 103,
 46: 104, 105, 106, 107, 108, 109, 110, 111,
 47: 112, 113, 114, 115, 116, 117, 118, 119,
 48: 120, 121, 122, 123, 124, 125, 126, 127,
 49: 128, 129, 130, 131, 132, 133, 134, 135,
 50: 136, 137, 138, 139, 140, 141, 142, 143,
 51: 144, 145, 146, 147, 148, 149, 150, 151,
 52: 152, 153, 154, 155, 156, 157, 158, 159,
 53: 160, 161, 162, 163, 164, 165, 166, 167,
 54: 168, 169, 170, 171, 172, 173, 174, 175,
 55: 176, 177, 178, 179, 180, 181, 182, 183,
 56: 184, 185, 186, 187, 188, 189, 190, 191,
 57: 192, 193, 194, 195, 196, 197, 198, 199,
 58: 200, 201, 202, 203, 204, 205, 206, 207,
 59: 208, 209, 210, 211, 212, 213, 214, 215,
 60: 216, 217, 218, 219, 220, 221, 222, 223,
 61: 224, 225, 226, 227, 228, 229, 230, 231,
 62: 232, 233, 234, 235, 236, 237, 238, 239,
 63: 240, 241, 242, 243, 244, 245, 246, 247,
 64: 248, 249, 250, 251, 252, 253, 254, 255,
 65:
 66: /* This table is a case flipping table. */
 67:
 68: 0, 1, 2, 3, 4, 5, 6, 7,
 69: 8, 9, 10, 11, 12, 13, 14, 15,
 70: 16, 17, 18, 19, 20, 21, 22, 23,
 71: 24, 25, 26, 27, 28, 29, 30, 31,
 72: 32, 33, 34, 35, 36, 37, 38, 39,
 73: 40, 41, 42, 43, 44, 45, 46, 47,
 74: 48, 49, 50, 51, 52, 53, 54, 55,
 75: 56, 57, 58, 59, 60, 61, 62, 63,
 76: 64, 97, 98, 99, 100, 101, 102, 103,
 77: 104, 105, 106, 107, 108, 109, 110, 111,
 78: 112, 113, 114, 115, 116, 117, 118, 119,
 79: 120, 121, 122, 91, 92, 93, 94, 95,
 80: 96, 65, 66, 67, 68, 69, 70, 71,

81: 72, 73, 74, 75, 76, 77, 78, 79,
 82: 80, 81, 82, 83, 84, 85, 86, 87,
 83: 88, 89, 90, 123, 124, 125, 126, 127,
 84: 128, 129, 130, 131, 132, 133, 134, 135,
 85: 136, 137, 138, 139, 140, 141, 142, 143,
 86: 144, 145, 146, 147, 148, 149, 150, 151,
 87: 152, 153, 154, 155, 156, 157, 158, 159,
 88: 160, 161, 162, 163, 164, 165, 166, 167,
 89: 168, 169, 170, 171, 172, 173, 174, 175,
 90: 176, 177, 178, 179, 180, 181, 182, 183,
 91: 184, 185, 186, 187, 188, 189, 190, 191,
 92: 192, 193, 194, 195, 196, 197, 198, 199,
 93: 200, 201, 202, 203, 204, 205, 206, 207,
 94: 208, 209, 210, 211, 212, 213, 214, 215,
 95: 216, 217, 218, 219, 220, 221, 222, 223,
 96: 224, 225, 226, 227, 228, 229, 230, 231,
 97: 232, 233, 234, 235, 236, 237, 238, 239,
 98: 240, 241, 242, 243, 244, 245, 246, 247,
 99: 248, 249, 250, 251, 252, 253, 254, 255,
 100:
 101: /* This table contains bit maps for various character classes. Each map is 32
 102: bytes long and the bits run from the least significant end of each byte. The
 103: classes that have their own maps are: space, xdigit, digit, upper, lower, word,
 104: graph, print, punct, and cntrl. Other classes are built from combinations. */
 105:
 106: 0x00, 0x3e, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
 107: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 108: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 109: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 110:
 111: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x03,
 112: 0x7e, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00,
 113: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 114: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 115:
 116: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x03,
 117: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 118: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 119: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
 120:
 121: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

122: 0xfe,0xff,0xff,0x07,0x00,0x00,0x00,0x00,
 123: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 124: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 125:
 126: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 127: 0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0x07,
 128: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 129: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 130:
 131: 0x00,0x00,0x00,0x00,0x00,0x00,0xff,0x03,
 132: 0xfe,0xff,0xff,0x87,0xfe,0xff,0xff,0x07,
 133: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 134: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 135:
 136: 0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,
 137: 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x7f,
 138: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 139: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 140:
 141: 0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,
 142: 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x7f,
 143: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 144: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 145:
 146: 0x00,0x00,0x00,0x00,0xfe,0xff,0x00,0xfc,
 147: 0x01,0x00,0x00,0xf8,0x01,0x00,0x00,0x78,
 148: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 149: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 150:
 151: 0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,
 152: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,
 153: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 154: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
 155:
 156: /* This table identifies various classes of character by individual bits:
 157: 0x01 white space character
 158: 0x02 letter
 159: 0x04 decimal digit
 160: 0x08 hexadecimal digit
 161: 0x10 alphanumeric or '_'
 162: 0x80 regular expression metacharacter or binary zero

```

163: */
164:
165: 0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 0- 7 */
166: 0x00,0x01,0x01,0x00,0x01,0x01,0x00,0x00, /* 8- 15 */
167: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 16- 23 */
168: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 24- 31 */
169: 0x01,0x00,0x00,0x00,0x80,0x00,0x00,0x00, /*  - ' */
170: 0x80,0x80,0x80,0x80,0x00,0x00,0x80,0x00, /* ( - / */
171: 0x1c,0x1c,0x1c,0x1c,0x1c,0x1c,0x1c,0x1c, /* 0 - 7 */
172: 0x1c,0x1c,0x00,0x00,0x00,0x00,0x00,0x80, /* 8 - ? */
173: 0x00,0x1a,0x1a,0x1a,0x1a,0x1a,0x1a,0x12, /* @ - G */
174: 0x12,0x12,0x12,0x12,0x12,0x12,0x12,0x12, /* H - O */
175: 0x12,0x12,0x12,0x12,0x12,0x12,0x12,0x12, /* P - W */
176: 0x12,0x12,0x12,0x80,0x80,0x00,0x80,0x10, /* X - _ */
177: 0x00,0x1a,0x1a,0x1a,0x1a,0x1a,0x1a,0x12, /* ` - g */
178: 0x12,0x12,0x12,0x12,0x12,0x12,0x12,0x12, /* h - o */
179: 0x12,0x12,0x12,0x12,0x12,0x12,0x12,0x12, /* p - w */
180: 0x12,0x12,0x12,0x80,0x80,0x00,0x00,0x00, /* x -127 */
181: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 128-135 */
182: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 136-143 */
183: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 144-151 */
184: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 152-159 */
185: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 160-167 */
186: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 168-175 */
187: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 176-183 */
188: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 184-191 */
189: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 192-199 */
190: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 200-207 */
191: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 208-215 */
192: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 216-223 */
193: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 224-231 */
194: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 232-239 */
195: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 240-247 */
196: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}; /* 248-255 */
197:
198: /* End of pcre_chartables.c */

```

File: sdm/VxWorks/libRegex/pcrecpp_internal.h

```
1: /*****
2: *   Perl-Compatible Regular Expressions   *
3: *****/
4:
5: /*
6: Copyright (c) 2005, Google Inc.
7: All rights reserved.
8:
9: -----
10: Redistribution and use in source and binary forms, with or without
11: modification, are permitted provided that the following conditions are met:
12:
13: * Redistributions of source code must retain the above copyright notice,
14:   this list of conditions and the following disclaimer.
15:
16: * Redistributions in binary form must reproduce the above copyright
17:   notice, this list of conditions and the following disclaimer in the
18:   documentation and/or other materials provided with the distribution.
19:
20: * Neither the name of the University of Cambridge nor the names of its
21:   contributors may be used to endorse or promote products derived from
22:   this software without specific prior written permission.
23:
24: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
25: "AS IS"
26: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
27: THE
28: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
29: PURPOSE
30: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
31: BE
32: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
33: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
34: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
35: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
36: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
37: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
38: POSSIBILITY OF SUCH DAMAGE.
39: -----
```

```

36: */
37:
38:
39: #ifndef PCRECPP_INTERNAL_H
40: #define PCRECPP_INTERNAL_H
41:
42: /* When compiling a DLL for Windows, the exported symbols have to be declared
43: using some MS magic. I found some useful information on this web page:
44: http://msdn2.microsoft.com/en-us/library/y4h7bcy6\(VS.80\).aspx. According to the
45: information there, using __declspec(dllexport) without "extern" we have a
46: definition; with "extern" we have a declaration. The settings here override the
47: setting in pcre.h. We use:
48:
49: PCRECPP_EXP_DECL    for declarations
50: PCRECPP_EXP_DEFN    for definitions of exported functions
51:
52: */
53:
54: #ifndef PCRECPP_EXP_DECL
55: #  ifdef _WIN32
56: #    ifndef PCRE_STATIC
57: #      define PCRECPP_EXP_DECL    extern __declspec(dllexport)
58: #      define PCRECPP_EXP_DEFN    __declspec(dllexport)
59: #    else
60: #      define PCRECPP_EXP_DECL    extern
61: #      define PCRECPP_EXP_DEFN
62: #    endif
63: #  else
64: #    define PCRECPP_EXP_DECL    extern
65: #    define PCRECPP_EXP_DEFN
66: #  endif
67: #endif
68:
69: #endif /* PCRECPP_INTERNAL_H */
70:
71: /* End of pcrecpp_internal.h */

```

File: sdm/VxWorks/libRegex/pcre_maketables.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:         Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```



```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_maketables(), which builds
42: character tables for PCRE in the current locale. The file is compiled on its
43: own as part of the PCRE library. However, it is also included in the
44: compilation of dftables.c, in which case the macro DFTABLES is defined. */
45:
46:
47: #ifndef DFTABLES
48: #  ifdef HAVE_CONFIG_H
49: #    include "config.h"
50: #  endif
51: #  include "pcre_internal.h"
52: #endif
53:
54:
55: /*****
56:  *      Create PCRE character tables      *
57:  *****/
58:
59: /* This function builds a set of character tables for use by PCRE and returns
60: a pointer to them. They are build using the ctype functions, and consequently
61: their contents will depend upon the current locale setting. When compiled as
62: part of the library, the store is obtained via pcre_malloc(), but when compiled
63: inside dftables, use malloc().
64:
65: Arguments:  none
66: Returns:   pointer to the contiguous block of data
67: */
68:
69: const unsigned char *
70: pcre_maketables(void)
71: {
72: unsigned char *yield, *p;
73: int i;
74:
75: #ifndef DFTABLES
76: yield = (unsigned char*)(pcre_malloc)(tables_length);

```

```

77: #else
78: yield = (unsigned char*)malloc(tables_length);
79: #endif
80:
81: if (yield == NULL) return NULL;
82: p = yield;
83:
84: /* First comes the lower casing table */
85:
86: for (i = 0; i < 256; i++) *p++ = tolower(i);
87:
88: /* Next the case-flipping table */
89:
90: for (i = 0; i < 256; i++) *p++ = islower(i)? toupper(i) : tolower(i);
91:
92: /* Then the character class tables. Don't try to be clever and save effort on
93: exclusive ones - in some locales things may be different. Note that the table
94: for "space" includes everything "isspace" gives, including VT in the default
95: locale. This makes it work for the POSIX class [:space:]. Note also that it is
96: possible for a character to be alnum or alpha without being lower or upper,
97: such as "male and female ordinals" ( \xAA and \xBA) in the fr_FR locale (at
98: least under Debian Linux's locales as of 12/2005). So we must test for alnum
99: specially. */
100:
101: memset(p, 0, cbit_length);
102: for (i = 0; i < 256; i++)
103: {
104: if (isdigit(i)) p[cbit_digit + i/8] |= 1 << (i&7);
105: if (isupper(i)) p[cbit_upper + i/8] |= 1 << (i&7);
106: if (islower(i)) p[cbit_lower + i/8] |= 1 << (i&7);
107: if (isalnum(i)) p[cbit_word + i/8] |= 1 << (i&7);
108: if (i == '_') p[cbit_word + i/8] |= 1 << (i&7);
109: if (isspace(i)) p[cbit_space + i/8] |= 1 << (i&7);
110: if (isxdigit(i)) p[cbit_xdigit + i/8] |= 1 << (i&7);
111: if (isgraph(i)) p[cbit_graph + i/8] |= 1 << (i&7);
112: if (isprint(i)) p[cbit_print + i/8] |= 1 << (i&7);
113: if (ispunct(i)) p[cbit_punct + i/8] |= 1 << (i&7);
114: if (iscntrl(i)) p[cbit_cntrl + i/8] |= 1 << (i&7);
115: }
116: p += cbit_length;
117:

```

```

118: /* Finally, the character type table. In this, we exclude VT from the white
119: space chars, because Perl doesn't recognize it as such for \s and for comments
120: within regexes. */
121:
122: for (i = 0; i < 256; i++)
123: {
124:     int x = 0;
125:     if (i != 0x0b && isspace(i)) x += ctype_space;
126:     if (isalpha(i)) x += ctype_letter;
127:     if (isdigit(i)) x += ctype_digit;
128:     if (isxdigit(i)) x += ctype_xdigit;
129:     if (isalnum(i) || i == '_') x += ctype_word;
130:
131:     /* Note: strchr includes the terminating zero in the characters it considers.
132:     In this instance, that is ok because we want binary zero to be flagged as a
133:     meta-character, which in this sense is any character that terminates a run
134:     of data characters. */
135:
136:     if (strchr("\\*+?{^$.|()[]", i) != 0) x += ctype_meta;
137:     *p++ = x;
138: }
139:
140: return yield;
141: }
142:
143: /* End of pcre_maketables.c */

```

File: sdm/VxWorks/libRegex/pcre_info.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:         Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_info(), which gives some
42: information about a compiled pattern. However, use of this function is now
43: deprecated, as it has been superseded by pcre_fullinfo(). */
44:
45:
46: #ifdef HAVE_CONFIG_H
47: #include "config.h"
48: #endif
49:
50: #include "pcre_internal.h"
51:
52:
53: /*****
54: * (Obsolete) Return info about compiled pattern *
55: *****/
56:
57: /* This is the original "info" function. It picks potentially useful data out
58: of the private structure, but its interface was too rigid. It remains for
59: backwards compatibility. The public options are passed back in an int - though
60: the re->options field has been expanded to a long int, all the public options
61: at the low end of it, and so even on 16-bit systems this will still be OK.
62: Therefore, I haven't changed the API for pcre_info().
63:
64: Arguments:
65: argument_re  points to compiled code
66: optptr      where to pass back the options
67: first_byte   where to pass back the first character,
68:              or -1 if multiline and all branches start ^,
69:              or -2 otherwise
70:
71: Returns:     number of capturing subpatterns
72:              or negative values on error
73: */
74:
75: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
76: pcre_info(const pcre *argument_re, int *optptr, int *first_byte)

```

```

77: {
78: real_pcre internal_re;
79: const real_pcre *re = (const real_pcre *)argument_re;
80: if (re == NULL) return PCRE_ERROR_NULL;
81: if (re->magic_number != MAGIC_NUMBER)
82: {
83: re = _pcre_try_flipped(re, &internal_re, NULL, NULL);
84: if (re == NULL) return PCRE_ERROR_BADMAGIC;
85: }
86: if (optptr != NULL) *optptr = (int)(re->options & PUBLIC_OPTIONS);
87: if (first_byte != NULL)
88: *first_byte = ((re->flags & PCRE_FIRSTSET) != 0)? re->first_byte :
89: ((re->flags & PCRE_STARTLINE) != 0)? -1 : -2;
90: return re->top_bracket;
91: }
92:
93: /* End of pcre_info.c */

```

File: sdm/VxWorks/libRegex/pcrecpparg.h

```
1: // Copyright (c) 2005, Google Inc.
2: // All rights reserved.
3: //
4: // Redistribution and use in source and binary forms, with or without
5: // modification, are permitted provided that the following conditions are
6: // met:
7: //
8: //   * Redistributions of source code must retain the above copyright
9: // notice, this list of conditions and the following disclaimer.
10: //   * Redistributions in binary form must reproduce the above
11: // copyright notice, this list of conditions and the following disclaimer
12: // in the documentation and/or other materials provided with the
13: // distribution.
14: //   * Neither the name of Google Inc. nor the names of its
15: // contributors may be used to endorse or promote products derived from
16: // this software without specific prior written permission.
17: //
18: // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
19: // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
20: // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
21: // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
22: // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
23: // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
24: // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
25: // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
26: // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27: // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
28: // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
29: //
30: // Author: Sanjay Ghemawat
31:
32: #ifndef _PCRECPPARG_H
33: #define _PCRECPPARG_H
34:
35: #include <stdlib.h> // for NULL
36: #include <string>
37:
38: #include <pcre.h>
39:
```

```

40: namespace pcrecpp {
41:
42: class StringPiece;
43:
44: // Hex/Octal/Binary?
45:
46: // Special class for parsing into objects that define a ParseFrom() method
47: template <class T>
48: class _RE_MatchObject {
49: public:
50: static inline bool Parse(const char* str, int n, void* dest) {
51:     if (dest == NULL) return true;
52:     T* object = reinterpret_cast<T*>(dest);
53:     return object->ParseFrom(str, n);
54: }
55: };
56:
57: class PCRECPP_EXP_DEFN Arg {
58: public:
59: // Empty constructor so we can declare arrays of Arg
60: Arg();
61:
62: // Constructor specially designed for NULL arguments
63: Arg(void*);
64:
65: typedef bool (*Parser)(const char* str, int n, void* dest);
66:
67: // Type-specific parsers
68: #define PCRE_MAKE_PARSER(type,name) \
69: Arg(type* p) : arg_(p), parser_(name) { } \
70: Arg(type* p, Parser parser) : arg_(p), parser_(parser) { }
71:
72:
73: PCRE_MAKE_PARSER(char, parse_char);
74: PCRE_MAKE_PARSER(unsigned char, parse_uchar);
75: PCRE_MAKE_PARSER(short, parse_short);
76: PCRE_MAKE_PARSER(unsigned short, parse_ushort);
77: PCRE_MAKE_PARSER(int, parse_int);
78: PCRE_MAKE_PARSER(unsigned int, parse_uint);
79: PCRE_MAKE_PARSER(long, parse_long);
80: PCRE_MAKE_PARSER(unsigned long, parse_ulong);

```



```

81: /*#if @pcre_have_long_long@*/
82: PCRE_MAKE_PARSER(long long,      parse_longlong);
83: /*#endif*/
84: /*#if @pcre_have_ulong_long@*/
85: PCRE_MAKE_PARSER(unsigned long long, parse_ulonglong);
86: /*#endif*/
87: PCRE_MAKE_PARSER(float,          parse_float);
88: PCRE_MAKE_PARSER(double,         parse_double);
89: PCRE_MAKE_PARSER(std::string,    parse_string);
90: PCRE_MAKE_PARSER(StringPiece,    parse_stringpiece);
91:
92: #undef PCRE_MAKE_PARSER
93:
94: // Generic constructor
95: template <class T> Arg(T*, Parser parser);
96: // Generic constructor template
97: template <class T> Arg(T* p)
98:   : arg_(p), parser_(RE_MatchObject<T>::Parse) {
99: }
100:
101: // Parse the data
102: bool Parse(const char* str, int n) const;
103:
104: private:
105: void*      arg_;
106: Parser     parser_;
107:
108: static bool parse_null      (const char* str, int n, void* dest);
109: static bool parse_char      (const char* str, int n, void* dest);
110: static bool parse_uchar     (const char* str, int n, void* dest);
111: static bool parse_float     (const char* str, int n, void* dest);
112: static bool parse_double    (const char* str, int n, void* dest);
113: static bool parse_string    (const char* str, int n, void* dest);
114: static bool parse_stringpiece (const char* str, int n, void* dest);
115:
116: #define PCRE_DECLARE_INTEGER_PARSER(name) \
117: private: \
118: static bool parse_ ## name (const char* str, int n, void* dest); \
119: static bool parse_ ## name ## _radix( \
120:   const char* str, int n, void* dest, int radix); \
121: public: \

```

```

122: static bool parse_ ## name ## _hex(const char* str, int n, void* dest); \
123: static bool parse_ ## name ## _octal(const char* str, int n, void* dest); \
124: static bool parse_ ## name ## _cradix(const char* str, int n, void* dest)
125:
126: PCRE_DECLARE_INTEGER_PARSER(short);
127: PCRE_DECLARE_INTEGER_PARSER(ushort);
128: PCRE_DECLARE_INTEGER_PARSER(int);
129: PCRE_DECLARE_INTEGER_PARSER(uint);
130: PCRE_DECLARE_INTEGER_PARSER(long);
131: PCRE_DECLARE_INTEGER_PARSER(ulong);
132: PCRE_DECLARE_INTEGER_PARSER(longlong);
133: PCRE_DECLARE_INTEGER_PARSER(ulonglong);
134:
135: #undef PCRE_DECLARE_INTEGER_PARSER
136: };
137:
138: inline Arg::Arg() : arg_(NULL), parser_(parse_null) { }
139: inline Arg::Arg(void* p) : arg_(p), parser_(parse_null) { }
140:
141: inline bool Arg::Parse(const char* str, int n) const {
142:     return (*parser_)(str, n, arg_);
143: }
144:
145: // This part of the parser, appropriate only for ints, deals with bases
146: #define MAKE_INTEGER_PARSER(type, name) \
147:     inline Arg Hex(type* ptr) { \
148:         return Arg(ptr, Arg::parse_ ## name ## _hex); } \
149:     inline Arg Octal(type* ptr) { \
150:         return Arg(ptr, Arg::parse_ ## name ## _octal); } \
151:     inline Arg CRadix(type* ptr) { \
152:         return Arg(ptr, Arg::parse_ ## name ## _cradix); }
153:
154: MAKE_INTEGER_PARSER(short,          short) /*          */
155: MAKE_INTEGER_PARSER(unsigned short,  ushort) /*          */
156: MAKE_INTEGER_PARSER(int,             int) /* Don't use semicolons */
157: MAKE_INTEGER_PARSER(unsigned int,    uint) /* after these statement */
158: MAKE_INTEGER_PARSER(long,            long) /* because they can cause */
159: MAKE_INTEGER_PARSER(unsigned long,   ulong) /* compiler warnings if */
160: /*#if @pcre_have_long_long@*/ /* the checking level is */
161: MAKE_INTEGER_PARSER(long long,       longlong) /* turned up high enough. */
162: /*#endif */ /*          */

```

```
163: /*#if @pcre_have_ulong_long@*/          /*          */
164: MAKE_INTEGER_PARSER(unsigned long long, ulonglong) /*          */
165: /*#endif*/
166:
167: #undef PCRE_IS_SET
168: #undef PCRE_SET_OR_CLEAR
169: #undef MAKE_INTEGER_PARSER
170:
171: } // namespace pcrecpp
172:
173:
174: #endif /* _PCRECPPARG_H */
```

File: sdm/VxWorks/libRegex/pcre_ord2utf8.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:         Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This file contains a private PCRE function that converts an ordinal
42: character value into a UTF8 string. */
43:
44: #ifdef HAVE_CONFIG_H
45: #include "config.h"
46: #endif
47:
48: #include "pcre_internal.h"
49:
50:
51: /*****
52:  *   Convert character value to UTF-8   *
53: *****/
54:
55: /* This function takes an integer value in the range 0 - 0x7fffffff
56: and encodes it as a UTF-8 character in 0 to 6 bytes.
57:
58: Arguments:
59:  cvalue   the character value
60:  buffer   pointer to buffer for result - at least 6 bytes long
61:
62: Returns:  number of characters placed in the buffer
63: */
64:
65: int
66: _pcre_ord2utf8(int cvalue, uschar *buffer)
67: {
68: #ifdef SUPPORT_UTF8
69: register int i, j;
70: for (i = 0; i < _pcre_utf8_table1_size; i++)
71: if (cvalue <= _pcre_utf8_table1[i]) break;
72: buffer += i;
73: for (j = i; j > 0; j--)
74: {
75: *buffer-- = 0x80 | (cvalue & 0x3f);
76: cvalue >>= 6;

```

```
77: }
78: *buffer = _pcre_utf8_table2[i] | cvalue;
79: return i + 1;
80: #else
81: (void)(cvalue); /* Keep compiler happy; this function won't ever be */
82: (void)(buffer); /* called when SUPPORT_UTF8 is not defined. */
83: return 0;
84: #endif
85: }
86:
87: /* End of pcre_ord2utf8.c */
```

File: sdm/VxWorks/libRegex/pcre_stringpiece.h

```
1: // Copyright (c) 2005, Google Inc.
2: // All rights reserved.
3: //
4: // Redistribution and use in source and binary forms, with or without
5: // modification, are permitted provided that the following conditions are
6: // met:
7: //
8: //   * Redistributions of source code must retain the above copyright
9: // notice, this list of conditions and the following disclaimer.
10: //   * Redistributions in binary form must reproduce the above
11: // copyright notice, this list of conditions and the following disclaimer
12: // in the documentation and/or other materials provided with the
13: // distribution.
14: //   * Neither the name of Google Inc. nor the names of its
15: // contributors may be used to endorse or promote products derived from
16: // this software without specific prior written permission.
17: //
18: // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
19: // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
20: // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
21: // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
22: // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
23: // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
24: // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
25: // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
26: // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27: // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
28: // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
29: //
30: // Author: Sanjay Ghemawat
31: //
32: // A string like object that points into another piece of memory.
33: // Useful for providing an interface that allows clients to easily
34: // pass in either a "const char*" or a "string".
35: //
36: // Argghh! I wish C++ literals were automatically of type "string".
37:
38: #ifndef _PCRE_STRINGPIECE_H
39: #define _PCRE_STRINGPIECE_H
```

```

40:
41: #include <string.h>
42: #include <string>
43: #include <iosfwd>    // for ostream forward-declaration
44:
45: #if pcre_have_type_traits
46: #define HAVE_TYPE_TRAITS
47: #include <type_traits.h>
48: #elif pcre_have_bits_type_traits
49: #define HAVE_TYPE_TRAITS
50: #include <bits/type_traits.h>
51: #endif
52:
53: #include <pcre.h>
54:
55: using std::string;
56:
57: namespace pcrecpp {
58:
59: class PCRECPP_EXP_DEFN StringPiece {
60: private:
61:     const char* ptr_;
62:     int length_;
63:
64: public:
65:     // We provide non-explicit singleton constructors so users can pass
66:     // in a "const char*" or a "string" wherever a "StringPiece" is
67:     // expected.
68:     StringPiece()
69:         : ptr_(NULL), length_(0) { }
70:     StringPiece(const char* str)
71:         : ptr_(str), length_(static_cast<int>(strlen(ptr_))) { }
72:     StringPiece(const unsigned char* str)
73:         : ptr_(reinterpret_cast<const char*>(str)),
74:         length_(static_cast<int>(strlen(ptr_))) { }
75:     StringPiece(const string& str)
76:         : ptr_(str.data()), length_(static_cast<int>(str.size())) { }
77:     StringPiece(const char* offset, int len)
78:         : ptr_(offset), length_(len) { }
79:
80:     // data() may return a pointer to a buffer with embedded NULs, and the

```



```

81: // returned buffer may or may not be null terminated. Therefore it is
82: // typically a mistake to pass data() to a routine that expects a NUL
83: // terminated string. Use "as_string().c_str()" if you really need to do
84: // this. Or better yet, change your routine so it does not rely on NUL
85: // termination.
86: const char* data() const { return ptr_; }
87: int size() const { return length_; }
88: bool empty() const { return length_ == 0; }
89:
90: void clear() { ptr_ = NULL; length_ = 0; }
91: void set(const char* buffer, int len) { ptr_ = buffer; length_ = len; }
92: void set(const char* str) {
93:     ptr_ = str;
94:     length_ = static_cast<int>(strlen(str));
95: }
96: void set(const void* buffer, int len) {
97:     ptr_ = reinterpret_cast<const char*>(buffer);
98:     length_ = len;
99: }
100:
101: char operator[](int i) const { return ptr_[i]; }
102:
103: void remove_prefix(int n) {
104:     ptr_ += n;
105:     length_ -= n;
106: }
107:
108: void remove_suffix(int n) {
109:     length_ -= n;
110: }
111:
112: bool operator==(const StringPiece& x) const {
113:     return ((length_ == x.length_) &&
114:            (memcmp(ptr_, x.ptr_, length_) == 0));
115: }
116: bool operator!=(const StringPiece& x) const {
117:     return !(*this == x);
118: }
119:
120: #define STRINGPIECE_BINARY_PREDICATE(cmp,auxcmp) \
121:     bool operator cmp (const StringPiece& x) const { \

```

```

122: int r = memcmp(ptr_, x.ptr_, length_ < x.length_ ? length_ : x.length_); \
123: return ((r auxcmp 0) || ((r == 0) && (length_ cmp x.length_))); \
124: }
125: STRINGPIECE_BINARY_PREDICATE(<, <);
126: STRINGPIECE_BINARY_PREDICATE(<=, <);
127: STRINGPIECE_BINARY_PREDICATE(>=, >);
128: STRINGPIECE_BINARY_PREDICATE(>, >);
129: #undef STRINGPIECE_BINARY_PREDICATE
130:
131: int compare(const StringPiece& x) const {
132: int r = memcmp(ptr_, x.ptr_, length_ < x.length_ ? length_ : x.length_);
133: if (r == 0) {
134:     if (length_ < x.length_) r = -1;
135:     else if (length_ > x.length_) r = +1;
136: }
137: return r;
138: }
139:
140: string as_string() const {
141: return string(data(), size());
142: }
143:
144: void CopyToString(string* target) const {
145: target->assign(ptr_, length_);
146: }
147:
148: // Does "this" start with "x"
149: bool starts_with(const StringPiece& x) const {
150: return ((length_ >= x.length_) && (memcmp(ptr_, x.ptr_, x.length_) == 0));
151: }
152: };
153:
154: } // namespace pcrecpp
155:
156: // -----
157: // Functions used to create STL containers that use StringPiece
158: // Remember that a StringPiece's lifetime had better be less than
159: // that of the underlying string or char*. If it is not, then you
160: // cannot safely store a StringPiece into an STL container
161: // -----
162:

```

```

163: #ifdef HAVE_TYPE_TRAITS
164: // This makes vector<StringPiece> really fast for some STL implementations
165: template<> struct __type_traits<pcrecpp::StringPiece> {
166:     typedef __true_type    has_trivial_default_constructor;
167:     typedef __true_type    has_trivial_copy_constructor;
168:     typedef __true_type    has_trivial_assignment_operator;
169:     typedef __true_type    has_trivial_destructor;
170:     typedef __true_type    is_POD_type;
171: };
172: #endif
173:
174: // allow StringPiece to be logged
175: std::ostream& operator<<(std::ostream& o, const pcrecpp::StringPiece& piece);
176:
177: #endif /* _PCRE_STRINGPIECE_H */

```

File: sdm/VxWorks/libRegex/pcre_config.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_config(). */
42:
43:
44: #ifdef HAVE_CONFIG_H
45: #include "config.h"
46: #endif
47:
48: #include "pcre_internal.h"
49:
50:
51: /*****
52:  * Return info about what features are configured *
53:  *****/
54:
55: /* This function has an extensible interface so that additional items can be
56: added compatibly.
57:
58: Arguments:
59:  what      what information is required
60:  where      where to put the information
61:
62: Returns:    0 if data returned, negative on error
63: */
64:
65: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
66: pcre_config(int what, void *where)
67: {
68: switch (what)
69: {
70: case PCRE_CONFIG_UTF8:
71: #ifdef SUPPORT_UTF8
72:  *((int *)where) = 1;
73: #else
74:  *((int *)where) = 0;
75: #endif
76: break;

```

```

77:
78: case PCRE_CONFIG_UNICODE_PROPERTIES:
79: #ifdef SUPPORT_UCP
80: *((int *)where) = 1;
81: #else
82: *((int *)where) = 0;
83: #endif
84: break;
85:
86: case PCRE_CONFIG_NEWLINE:
87: *((int *)where) = NEWLINE;
88: break;
89:
90: case PCRE_CONFIG_BSR:
91: #ifdef BSR_ANYCRLF
92: *((int *)where) = 1;
93: #else
94: *((int *)where) = 0;
95: #endif
96: break;
97:
98: case PCRE_CONFIG_LINK_SIZE:
99: *((int *)where) = LINK_SIZE;
100: break;
101:
102: case PCRE_CONFIG_POSIX_MALLOC_THRESHOLD:
103: *((int *)where) = POSIX_MALLOC_THRESHOLD;
104: break;
105:
106: case PCRE_CONFIG_MATCH_LIMIT:
107: *((unsigned int *)where) = MATCH_LIMIT;
108: break;
109:
110: case PCRE_CONFIG_MATCH_LIMIT_RECURSION:
111: *((unsigned int *)where) = MATCH_LIMIT_RECURSION;
112: break;
113:
114: case PCRE_CONFIG_STACKRECURSE:
115: #ifdef NO_RECURSE
116: *((int *)where) = 0;
117: #else

```

```
118: *((int *)where) = 1;
119: #endif
120: break;
121:
122: default: return PCRE_ERROR_BADOPTION;
123: }
124:
125: return 0;
126: }
127:
128: /* End of pcre_config.c */
```

File: sdm/VxWorks/libRegex/pcrecpp.cc

```
1: // Copyright (c) 2005, Google Inc.
2: // All rights reserved.
3: //
4: // Redistribution and use in source and binary forms, with or without
5: // modification, are permitted provided that the following conditions are
6: // met:
7: //
8: //   * Redistributions of source code must retain the above copyright
9: // notice, this list of conditions and the following disclaimer.
10: //   * Redistributions in binary form must reproduce the above
11: // copyright notice, this list of conditions and the following disclaimer
12: // in the documentation and/or other materials provided with the
13: // distribution.
14: //   * Neither the name of Google Inc. nor the names of its
15: // contributors may be used to endorse or promote products derived from
16: // this software without specific prior written permission.
17: //
18: // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
19: // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
20: // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
21: // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
22: // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
23: // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
24: // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
25: // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
26: // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27: // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
28: // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
29: //
30: // Author: Sanjay Ghemawat
31:
32: #ifdef HAVE_CONFIG_H
33: #include "config.h"
34: #endif
35:
36: #include <stdlib.h>
37: #include <stdio.h>
38: #include <ctype.h>
39: #include <limits.h>    /* for SHRT_MIN, USHRT_MAX, etc */
```



```

40: #include <assert.h>
41: #include <errno.h>
42: #include <string>
43: #include <algorithm>
44:
45: #include "pcrecpp_internal.h"
46: #include "pcre.h"
47: #include "pcrecpp.h"
48: #include "pcre_stringpiece.h"
49:
50:
51: namespace pcrecpp {
52:
53: // Maximum number of args we can set
54: static const int kMaxArgs = 16;
55: static const int kVecSize = (1 + kMaxArgs) * 3; // results + PCRE workspace
56:
57: // Special object that stands-in for no argument
58: Arg RE::no_arg((void*)NULL);
59:
60: // This is for ABI compatibility with old versions of pcre (pre-7.6),
61: // which defined a global no_arg variable instead of putting it in the
62: // RE class. This works on GCC >= 3, at least. It definitely works
63: // for ELF, but may not for other object formats (Mach-O, for
64: // instance, does not support aliases.) We could probably have a more
65: // inclusive test if we ever needed it. (Note that not only the
66: // __attribute__ syntax, but also __USER_LABEL_PREFIX__, are
67: // gnu-specific.)
68: #if defined(__GNUC__) && __GNUC__ >= 3 && defined(__ELF__)
69: # define ULP_AS_STRING(x)      ULP_AS_STRING_INTERNAL(x)
70: # define ULP_AS_STRING_INTERNAL(x) #x
71: # define USER_LABEL_PREFIX_STR  ULP_AS_STRING(__USER_LABEL_PREFIX__)
72: extern Arg no_arg
73: __attribute__((alias(USER_LABEL_PREFIX_STR "_ZN7pcrecpp2RE6no_argE")));
74: #endif
75:
76: // If a regular expression has no error, its error_ field points here
77: static const string empty_string;
78:
79: // If the user doesn't ask for any options, we just use this one
80: static RE_Options default_options;

```

```

81:
82: void RE::Init(const string& pat, const RE_Options* options) {
83:     pattern_ = pat;
84:     if (options == NULL) {
85:         options_ = default_options;
86:     } else {
87:         options_ = *options;
88:     }
89:     error_ = &empty_string;
90:     re_full_ = NULL;
91:     re_partial_ = NULL;
92:
93:     re_partial_ = Compile(UNANCHORED);
94:     if (re_partial_ != NULL) {
95:         re_full_ = Compile(ANCHOR_BOTH);
96:     }
97: }
98:
99: void RE::Cleanup() {
100:     if (re_full_ != NULL)      (*pcre_free)(re_full_);
101:     if (re_partial_ != NULL)  (*pcre_free)(re_partial_);
102:     if (error_ != &empty_string) delete error_;
103: }
104:
105:
106: RE::~~RE() {
107:     Cleanup();
108: }
109:
110:
111: pcre* RE::Compile(Anchor anchor) {
112:     // First, convert RE_Options into pcre options
113:     int pcre_options = 0;
114:     pcre_options = options_.all_options();
115:
116:     // Special treatment for anchoring. This is needed because at
117:     // runtime pcre only provides an option for anchoring at the
118:     // beginning of a string (unless you use offset).
119:     //
120:     // There are three types of anchoring we want:
121:     //  UNANCHORED    Compile the original pattern, and use

```

```

122: //          a pcre unanchored match.
123: //  ANCHOR_START  Compile the original pattern, and use
124: //          a pcre anchored match.
125: //  ANCHOR_BOTH   Tack a "\z" to the end of the original pattern
126: //          and use a pcre anchored match.
127:
128: const char* compile_error;
129: int eoffset;
130: pcre* re;
131: if (anchor != ANCHOR_BOTH) {
132:     re = pcre_compile(pattern_.c_str(), pcre_options,
133:                       &compile_error, &eoffset, NULL);
134: } else {
135:     // Tack a '\z' at the end of RE.  Parenthesize it first so that
136:     // the '\z' applies to all top-level alternatives in the regexp.
137:     string wrapped = "(?:"; // A non-counting grouping operator
138:     wrapped += pattern_;
139:     wrapped += ") \z";
140:     re = pcre_compile(wrapped.c_str(), pcre_options,
141:                       &compile_error, &eoffset, NULL);
142: }
143: if (re == NULL) {
144:     if (error_ == &empty_string) error_ = new string(compile_error);
145: }
146: return re;
147: }
148:
149: /***** Matching interfaces *****/
150:
151: bool RE::FullMatch(const StringPiece& text,
152:                    const Arg& ptr1,
153:                    const Arg& ptr2,
154:                    const Arg& ptr3,
155:                    const Arg& ptr4,
156:                    const Arg& ptr5,
157:                    const Arg& ptr6,
158:                    const Arg& ptr7,
159:                    const Arg& ptr8,
160:                    const Arg& ptr9,
161:                    const Arg& ptr10,
162:                    const Arg& ptr11,

```

```

163:         const Arg& ptr12,
164:         const Arg& ptr13,
165:         const Arg& ptr14,
166:         const Arg& ptr15,
167:         const Arg& ptr16) const {
168:     const Arg* args[kMaxArgs];
169:     int n = 0;
170:     if (&ptr1 == &no_arg) goto done; args[n++] = &ptr1;
171:     if (&ptr2 == &no_arg) goto done; args[n++] = &ptr2;
172:     if (&ptr3 == &no_arg) goto done; args[n++] = &ptr3;
173:     if (&ptr4 == &no_arg) goto done; args[n++] = &ptr4;
174:     if (&ptr5 == &no_arg) goto done; args[n++] = &ptr5;
175:     if (&ptr6 == &no_arg) goto done; args[n++] = &ptr6;
176:     if (&ptr7 == &no_arg) goto done; args[n++] = &ptr7;
177:     if (&ptr8 == &no_arg) goto done; args[n++] = &ptr8;
178:     if (&ptr9 == &no_arg) goto done; args[n++] = &ptr9;
179:     if (&ptr10 == &no_arg) goto done; args[n++] = &ptr10;
180:     if (&ptr11 == &no_arg) goto done; args[n++] = &ptr11;
181:     if (&ptr12 == &no_arg) goto done; args[n++] = &ptr12;
182:     if (&ptr13 == &no_arg) goto done; args[n++] = &ptr13;
183:     if (&ptr14 == &no_arg) goto done; args[n++] = &ptr14;
184:     if (&ptr15 == &no_arg) goto done; args[n++] = &ptr15;
185:     if (&ptr16 == &no_arg) goto done; args[n++] = &ptr16;
186:     done:
187:
188:     int consumed;
189:     int vec[kVecSize];
190:     return DoMatchImpl(text, ANCHOR_BOTH, &consumed, args, n, vec, kVecSize);
191: }
192:
193: bool RE::PartialMatch(const StringPiece& text,
194:         const Arg& ptr1,
195:         const Arg& ptr2,
196:         const Arg& ptr3,
197:         const Arg& ptr4,
198:         const Arg& ptr5,
199:         const Arg& ptr6,
200:         const Arg& ptr7,
201:         const Arg& ptr8,
202:         const Arg& ptr9,
203:         const Arg& ptr10,

```

```

204:         const Arg& ptr11,
205:         const Arg& ptr12,
206:         const Arg& ptr13,
207:         const Arg& ptr14,
208:         const Arg& ptr15,
209:         const Arg& ptr16) const {
210:     const Arg* args[kMaxArgs];
211:     int n = 0;
212:     if (&ptr1 == &no_arg) goto done; args[n++] = &ptr1;
213:     if (&ptr2 == &no_arg) goto done; args[n++] = &ptr2;
214:     if (&ptr3 == &no_arg) goto done; args[n++] = &ptr3;
215:     if (&ptr4 == &no_arg) goto done; args[n++] = &ptr4;
216:     if (&ptr5 == &no_arg) goto done; args[n++] = &ptr5;
217:     if (&ptr6 == &no_arg) goto done; args[n++] = &ptr6;
218:     if (&ptr7 == &no_arg) goto done; args[n++] = &ptr7;
219:     if (&ptr8 == &no_arg) goto done; args[n++] = &ptr8;
220:     if (&ptr9 == &no_arg) goto done; args[n++] = &ptr9;
221:     if (&ptr10 == &no_arg) goto done; args[n++] = &ptr10;
222:     if (&ptr11 == &no_arg) goto done; args[n++] = &ptr11;
223:     if (&ptr12 == &no_arg) goto done; args[n++] = &ptr12;
224:     if (&ptr13 == &no_arg) goto done; args[n++] = &ptr13;
225:     if (&ptr14 == &no_arg) goto done; args[n++] = &ptr14;
226:     if (&ptr15 == &no_arg) goto done; args[n++] = &ptr15;
227:     if (&ptr16 == &no_arg) goto done; args[n++] = &ptr16;
228:     done:
229:
230:     int consumed;
231:     int vec[kVecSize];
232:     return DoMatchImpl(text, UNANCHORED, &consumed, args, n, vec, kVecSize);
233: }
234:
235: bool RE::Consume(StringPiece* input,
236:         const Arg& ptr1,
237:         const Arg& ptr2,
238:         const Arg& ptr3,
239:         const Arg& ptr4,
240:         const Arg& ptr5,
241:         const Arg& ptr6,
242:         const Arg& ptr7,
243:         const Arg& ptr8,
244:         const Arg& ptr9,

```

```

245:         const Arg& ptr10,
246:         const Arg& ptr11,
247:         const Arg& ptr12,
248:         const Arg& ptr13,
249:         const Arg& ptr14,
250:         const Arg& ptr15,
251:         const Arg& ptr16) const {
252:     const Arg* args[kMaxArgs];
253:     int n = 0;
254:     if (&ptr1 == &no_arg) goto done; args[n++] = &ptr1;
255:     if (&ptr2 == &no_arg) goto done; args[n++] = &ptr2;
256:     if (&ptr3 == &no_arg) goto done; args[n++] = &ptr3;
257:     if (&ptr4 == &no_arg) goto done; args[n++] = &ptr4;
258:     if (&ptr5 == &no_arg) goto done; args[n++] = &ptr5;
259:     if (&ptr6 == &no_arg) goto done; args[n++] = &ptr6;
260:     if (&ptr7 == &no_arg) goto done; args[n++] = &ptr7;
261:     if (&ptr8 == &no_arg) goto done; args[n++] = &ptr8;
262:     if (&ptr9 == &no_arg) goto done; args[n++] = &ptr9;
263:     if (&ptr10 == &no_arg) goto done; args[n++] = &ptr10;
264:     if (&ptr11 == &no_arg) goto done; args[n++] = &ptr11;
265:     if (&ptr12 == &no_arg) goto done; args[n++] = &ptr12;
266:     if (&ptr13 == &no_arg) goto done; args[n++] = &ptr13;
267:     if (&ptr14 == &no_arg) goto done; args[n++] = &ptr14;
268:     if (&ptr15 == &no_arg) goto done; args[n++] = &ptr15;
269:     if (&ptr16 == &no_arg) goto done; args[n++] = &ptr16;
270:     done:
271:
272:     int consumed;
273:     int vec[kVecSize];
274:     if (DoMatchImpl(*input, ANCHOR_START, &consumed,
275:         args, n, vec, kVecSize)) {
276:         input->remove_prefix(consumed);
277:         return true;
278:     } else {
279:         return false;
280:     }
281: }
282:
283: bool RE::FindAndConsume(StringPiece* input,
284:         const Arg& ptr1,
285:         const Arg& ptr2,

```

```

286:         const Arg& ptr3,
287:         const Arg& ptr4,
288:         const Arg& ptr5,
289:         const Arg& ptr6,
290:         const Arg& ptr7,
291:         const Arg& ptr8,
292:         const Arg& ptr9,
293:         const Arg& ptr10,
294:         const Arg& ptr11,
295:         const Arg& ptr12,
296:         const Arg& ptr13,
297:         const Arg& ptr14,
298:         const Arg& ptr15,
299:         const Arg& ptr16) const {
300:     const Arg* args[kMaxArgs];
301:     int n = 0;
302:     if (&ptr1 == &no_arg) goto done; args[n++] = &ptr1;
303:     if (&ptr2 == &no_arg) goto done; args[n++] = &ptr2;
304:     if (&ptr3 == &no_arg) goto done; args[n++] = &ptr3;
305:     if (&ptr4 == &no_arg) goto done; args[n++] = &ptr4;
306:     if (&ptr5 == &no_arg) goto done; args[n++] = &ptr5;
307:     if (&ptr6 == &no_arg) goto done; args[n++] = &ptr6;
308:     if (&ptr7 == &no_arg) goto done; args[n++] = &ptr7;
309:     if (&ptr8 == &no_arg) goto done; args[n++] = &ptr8;
310:     if (&ptr9 == &no_arg) goto done; args[n++] = &ptr9;
311:     if (&ptr10 == &no_arg) goto done; args[n++] = &ptr10;
312:     if (&ptr11 == &no_arg) goto done; args[n++] = &ptr11;
313:     if (&ptr12 == &no_arg) goto done; args[n++] = &ptr12;
314:     if (&ptr13 == &no_arg) goto done; args[n++] = &ptr13;
315:     if (&ptr14 == &no_arg) goto done; args[n++] = &ptr14;
316:     if (&ptr15 == &no_arg) goto done; args[n++] = &ptr15;
317:     if (&ptr16 == &no_arg) goto done; args[n++] = &ptr16;
318: done:
319:
320:     int consumed;
321:     int vec[kVecSize];
322:     if (DoMatchImpl(*input, UNANCHORED, &consumed,
323:         args, n, vec, kVecSize)) {
324:         input->remove_prefix(consumed);
325:         return true;
326:     } else {

```

```

327:   return false;
328: }
329: }
330:
331: bool RE::Replace(const StringPiece& rewrite,
332:                 string *str) const {
333:   int vec[kVecSize];
334:   int matches = TryMatch(*str, 0, UNANCHORED, vec, kVecSize);
335:   if (matches == 0)
336:     return false;
337:
338:   string s;
339:   if (!Rewrite(&s, rewrite, *str, vec, matches))
340:     return false;
341:
342:   assert(vec[0] >= 0);
343:   assert(vec[1] >= 0);
344:   str->replace(vec[0], vec[1] - vec[0], s);
345:   return true;
346: }
347:
348: // Returns PCRE_NEWLINE_CRLF, PCRE_NEWLINE_CR, or PCRE_NEWLINE_LF.
349: // Note that PCRE_NEWLINE_CRLF is defined to be P_N_CR | P_N_LF.
350: // Modified by PH to add PCRE_NEWLINE_ANY and PCRE_NEWLINE_ANYCRLF.
351:
352: static int NewlineMode(int pcre_options) {
353:   // TODO: if we can make it threadsafe, cache this var
354:   int newline_mode = 0;
355:   /* if (newline_mode) return newline_mode; */ // do this once it's cached
356:   if (pcre_options & (PCRE_NEWLINE_CRLF|PCRE_NEWLINE_CR|PCRE_NEWLINE_LF|
357:                       PCRE_NEWLINE_ANY|PCRE_NEWLINE_ANYCRLF)) {
358:     newline_mode = (pcre_options &
359:                    (PCRE_NEWLINE_CRLF|PCRE_NEWLINE_CR|PCRE_NEWLINE_LF|
360:                     PCRE_NEWLINE_ANY|PCRE_NEWLINE_ANYCRLF));
361:   } else {
362:     int newline;
363:     pcre_config(PCRE_CONFIG_NEWLINE, &newline);
364:     if (newline == 10)
365:       newline_mode = PCRE_NEWLINE_LF;
366:     else if (newline == 13)
367:       newline_mode = PCRE_NEWLINE_CR;

```



```

368:     else if (newline == 3338)
369:         newline_mode = PCRE_NEWLINE_CRLF;
370:     else if (newline == -1)
371:         newline_mode = PCRE_NEWLINE_ANY;
372:     else if (newline == -2)
373:         newline_mode = PCRE_NEWLINE_ANYCRLF;
374:     else
375:         assert(NULL == "Unexpected return value from pcre_config(NEWLINE)");
376: }
377: return newline_mode;
378: }
379:
380: int RE::GlobalReplace(const StringPiece& rewrite,
381:     string *str) const {
382:     int count = 0;
383:     int vec[kVecSize];
384:     string out;
385:     int start = 0;
386:     int lastend = -1;
387:
388:     while (start <= static_cast<int>(str->length())) {
389:         int matches = TryMatch(*str, start, UNANCHORED, vec, kVecSize);
390:         if (matches <= 0)
391:             break;
392:         int matchstart = vec[0], matchend = vec[1];
393:         assert(matchstart >= start);
394:         assert(matchend >= matchstart);
395:         if (matchstart == matchend && matchstart == lastend) {
396:             // advance one character if we matched an empty string at the same
397:             // place as the last match occurred
398:             matchend = start + 1;
399:             // If the current char is CR and we're in CRLF mode, skip LF too.
400:             // Note it's better to call pcre_fullinfo() than to examine
401:             // all_options(), since options_ could have changed bewteen
402:             // compile-time and now, but this is simpler and safe enough.
403:             // Modified by PH to add ANY and ANYCRLF.
404:             if (start+1 < static_cast<int>(str->length()) &&
405:                 (*str)[start] == '\r' && (*str)[start+1] == '\n' &&
406:                 NewlineMode(options_.all_options()) == PCRE_NEWLINE_CRLF ||
407:                 NewlineMode(options_.all_options()) == PCRE_NEWLINE_ANY ||
408:                 NewlineMode(options_.all_options()) == PCRE_NEWLINE_ANYCRLF)

```

```

409:     ) {
410:         matchend++;
411:     }
412:     // We also need to advance more than one char if we're in utf8 mode.
413: #ifdef SUPPORT_UTF8
414:     if (options_.utf8()) {
415:         while (matchend < static_cast<int>(str->length()) &&
416:             ((*str)[matchend] & 0xc0) == 0x80)
417:             matchend++;
418:     }
419: #endif
420:     if (matchend <= static_cast<int>(str->length()))
421:         out.append(*str, start, matchend - start);
422:     start = matchend;
423: } else {
424:     out.append(*str, start, matchstart - start);
425:     Rewrite(&out, rewrite, *str, vec, matches);
426:     start = matchend;
427:     lastend = matchend;
428:     count++;
429: }
430: }
431:
432: if (count == 0)
433:     return 0;
434:
435: if (start < static_cast<int>(str->length()))
436:     out.append(*str, start, str->length() - start);
437: swap(out, *str);
438: return count;
439: }
440:
441: bool RE::Extract(const StringPiece& rewrite,
442:                 const StringPiece& text,
443:                 string *out) const {
444:     int vec[kVecSize];
445:     int matches = TryMatch(text, 0, UNANCHORED, vec, kVecSize);
446:     if (matches == 0)
447:         return false;
448:     out->erase();
449:     return Rewrite(out, rewrite, text, vec, matches);

```

```

450: }
451:
452: /*static*/ string RE::QuoteMeta(const StringPiece& unquoted) {
453:     string result;
454:
455:     // Escape any ascii character not in [A-Za-z_0-9].
456:     //
457:     // Note that it's legal to escape a character even if it has no
458:     // special meaning in a regular expression -- so this function does
459:     // that. (This also makes it identical to the perl function of the
460:     // same name; see `perldoc -f quotemeta`.) The one exception is
461:     // escaping NUL: rather than doing backslash + NUL, like perl does,
462:     // we do '\0', because pcre itself doesn't take embedded NUL chars.
463:     for (int ii = 0; ii < unquoted.size(); ++ii) {
464:         // Note that using 'isalnum' here raises the benchmark time from
465:         // 32ns to 58ns:
466:         if (unquoted[ii] == '\0') {
467:             result += "\\0";
468:         } else if ((unquoted[ii] < 'a' || unquoted[ii] > 'z') &&
469:                    (unquoted[ii] < 'A' || unquoted[ii] > 'Z') &&
470:                    (unquoted[ii] < '0' || unquoted[ii] > '9') &&
471:                    unquoted[ii] != '_' &&
472:                    // If this is the part of a UTF8 or Latin1 character, we need
473:                    // to copy this byte without escaping. Experimentally this is
474:                    // what works correctly with the regexp library.
475:                    !(unquoted[ii] & 128)) {
476:             result += '\\';
477:             result += unquoted[ii];
478:         } else {
479:             result += unquoted[ii];
480:         }
481:     }
482:
483:     return result;
484: }
485:
486: /***** Actual matching and rewriting code *****/
487:
488: int RE::TryMatch(const StringPiece& text,
489:                 int startpos,
490:                 Anchor anchor,

```

```

491:         int *vec,
492:         int vecsize) const {
493:     pcre* re = (anchor == ANCHOR_BOTH) ? re_full_ : re_partial_;
494:     if (re == NULL) {
495:         //fprintf(stderr, "Matching against invalid re: %s \n", error_>c_str());
496:         return 0;
497:     }
498:
499:     pcre_extra extra = { 0, 0, 0, 0, 0, 0 };
500:     if (options_.match_limit() > 0) {
501:         extra.flags |= PCRE_EXTRA_MATCH_LIMIT;
502:         extra.match_limit = options_.match_limit();
503:     }
504:     if (options_.match_limit_recursion() > 0) {
505:         extra.flags |= PCRE_EXTRA_MATCH_LIMIT_RECURSION;
506:         extra.match_limit_recursion = options_.match_limit_recursion();
507:     }
508:     int rc = pcre_exec(re,          // The regular expression object
509:                       &extra,
510:                       (text.data() == NULL) ? "" : text.data(),
511:                       text.size(),
512:                       startpos,
513:                       (anchor == UNANCHORED) ? 0 : PCRE_ANCHORED,
514:                       vec,
515:                       vecsize);
516:
517:     // Handle errors
518:     if (rc == PCRE_ERROR_NOMATCH) {
519:         return 0;
520:     } else if (rc < 0) {
521:         //fprintf(stderr, "Unexpected return code: %d when matching '%s' \n",
522:         //    re, pattern_.c_str());
523:         return 0;
524:     } else if (rc == 0) {
525:         // pcre_exec() returns 0 as a special case when the number of
526:         // capturing subpatterns exceeds the size of the vector.
527:         // When this happens, there is a match and the output vector
528:         // is filled, but we miss out on the positions of the extra subpatterns.
529:         rc = vecsize / 2;
530:     }
531:

```

```

532: return rc;
533: }
534:
535: bool RE::DoMatchImpl(const StringPiece& text,
536:                     Anchor anchor,
537:                     int* consumed,
538:                     const Arg* const* args,
539:                     int n,
540:                     int* vec,
541:                     int vecsize) const {
542: assert((1 + n) * 3 <= vecsize); // results + PCRE workspace
543: int matches = TryMatch(text, 0, anchor, vec, vecsize);
544: assert(matches >= 0); // TryMatch never returns negatives
545: if (matches == 0)
546:     return false;
547:
548: *consumed = vec[1];
549:
550: if (n == 0 || args == NULL) {
551:     // We are not interested in results
552:     return true;
553: }
554:
555: if (NumberOfCapturingGroups() < n) {
556:     // RE has fewer capturing groups than number of arg pointers passed in
557:     return false;
558: }
559:
560: // If we got here, we must have matched the whole pattern.
561: // We do not need (can not do) any more checks on the value of 'matches' here
562: // -- see the comment for TryMatch.
563: for (int i = 0; i < n; i++) {
564:     const int start = vec[2*(i+1)];
565:     const int limit = vec[2*(i+1)+1];
566:     if (!args[i]->Parse(text.data() + start, limit-start)) {
567:         // TODO: Should we indicate what the error was?
568:         return false;
569:     }
570: }
571:
572: return true;

```

```

573: }
574:
575: bool RE::DoMatch(const StringPiece& text,
576:                 Anchor anchor,
577:                 int* consumed,
578:                 const Arg* const args[],
579:                 int n) const {
580:     assert(n >= 0);
581:     size_t const vecsize = (1 + n) * 3; // results + PCRE workspace
582:                                     // (as for kVecSize)
583:     int space[21]; // use stack allocation for small vecsize (common case)
584:     int* vec = vecsize <= 21 ? space : new int[vecsize];
585:     bool retval = DoMatchImpl(text, anchor, consumed, args, n, vec, vecsize);
586:     if (vec != space) delete [] vec;
587:     return retval;
588: }
589:
590: bool RE::Rewrite(string *out, const StringPiece &rewrite,
591:                  const StringPiece &text, int *vec, int veclen) const {
592:     for (const char *s = rewrite.data(), *end = s + rewrite.size();
593:          s < end; s++) {
594:         int c = *s;
595:         if (c == '\\') {
596:             c = *++s;
597:             if (isdigit(c)) {
598:                 int n = (c - '0');
599:                 if (n >= veclen) {
600:                     //fprintf(stderr, requested group %d in regexp %.*s \n",
601:                     //      n, rewrite.size(), rewrite.data());
602:                     return false;
603:                 }
604:                 int start = vec[2 * n];
605:                 if (start >= 0)
606:                     out->append(text.data() + start, vec[2 * n + 1] - start);
607:             } else if (c == '\\') {
608:                 *out += '\\';
609:             } else {
610:                 //fprintf(stderr, "invalid rewrite pattern: %.*s \n",
611:                 //      rewrite.size(), rewrite.data());
612:                 return false;
613:             }

```

```

614: } else {
615:     *out += c;
616: }
617: }
618: return true;
619: }
620:
621: // Return the number of capturing subpatterns, or -1 if the
622: // regexp wasn't valid on construction.
623: int RE::NumberOfCapturingGroups() const {
624:     if (re_partial_ == NULL) return -1;
625:
626:     int result;
627:     int pcre_retval = pcre_fullinfo(re_partial_, // The regular expression object
628:                                     NULL,        // We did not study the pattern
629:                                     PCRE_INFO_CAPTURECOUNT,
630:                                     &result);
631:     assert(pcre_retval == 0);
632:     return result;
633: }
634:
635: /***** Parsers for various types *****/
636:
637: bool Arg::parse_null(const char* str, int n, void* dest) {
638:     // We fail if somebody asked us to store into a non-NULL void* pointer
639:     return (dest == NULL);
640: }
641:
642: bool Arg::parse_string(const char* str, int n, void* dest) {
643:     if (dest == NULL) return true;
644:     reinterpret_cast<string*>(dest)->assign(str, n);
645:     return true;
646: }
647:
648: bool Arg::parse_stringpiece(const char* str, int n, void* dest) {
649:     if (dest == NULL) return true;
650:     reinterpret_cast<StringPiece*>(dest)->set(str, n);
651:     return true;
652: }
653:
654: bool Arg::parse_char(const char* str, int n, void* dest) {

```

```

655: if (n != 1) return false;
656: if (dest == NULL) return true;
657: *(reinterpret_cast<char*>(dest)) = str[0];
658: return true;
659: }
660:
661: bool Arg::parse_uchar(const char* str, int n, void* dest) {
662: if (n != 1) return false;
663: if (dest == NULL) return true;
664: *(reinterpret_cast<unsigned char*>(dest)) = str[0];
665: return true;
666: }
667:
668: // Largest number spec that we are willing to parse
669: static const int kMaxNumberLength = 32;
670:
671: // REQUIRES "buf" must have length at least kMaxNumberLength+1
672: // REQUIRES "n > 0"
673: // Copies "str" into "buf" and null-terminates if necessary.
674: // Returns one of:
675: //   a. "str" if no termination is needed
676: //   b. "buf" if the string was copied and null-terminated
677: //   c. "" if the input was invalid and has no hope of being parsed
678: static const char* TerminateNumber(char* buf, const char* str, int n) {
679: if ((n > 0) && isspace(*str)) {
680: // We are less forgiving than the strtoux() routines and do not
681: // allow leading spaces.
682: return "";
683: }
684:
685: // See if the character right after the input text may potentially
686: // look like a digit.
687: if (isdigit(str[n]) ||
688:     ((str[n] >= 'a') && (str[n] <= 'f')) ||
689:     ((str[n] >= 'A') && (str[n] <= 'F')))) {
690: if (n > kMaxNumberLength) return ""; // Input too big to be a valid number
691: memcpy(buf, str, n);
692: buf[n] = '\0';
693: return buf;
694: } else {
695: // We can parse right out of the supplied string, so return it.

```



```

696:   return str;
697: }
698: }
699:
700: bool Arg::parse_long_radix(const char* str,
701:                           int n,
702:                           void* dest,
703:                           int radix) {
704:   if (n == 0) return false;
705:   char buf[kMaxNumberLength+1];
706:   str = TerminateNumber(buf, str, n);
707:   char* end;
708:   errno = 0;
709:   long r = strtol(str, &end, radix);
710:   if (end != str + n) return false; // Leftover junk
711:   if (errno) return false;
712:   if (dest == NULL) return true;
713:   *(reinterpret_cast<long*>(dest)) = r;
714:   return true;
715: }
716:
717: bool Arg::parse_ulong_radix(const char* str,
718:                             int n,
719:                             void* dest,
720:                             int radix) {
721:   if (n == 0) return false;
722:   char buf[kMaxNumberLength+1];
723:   str = TerminateNumber(buf, str, n);
724:   if (str[0] == '-') return false; // strtoul() on a negative number?!
725:   char* end;
726:   errno = 0;
727:   unsigned long r = strtoul(str, &end, radix);
728:   if (end != str + n) return false; // Leftover junk
729:   if (errno) return false;
730:   if (dest == NULL) return true;
731:   *(reinterpret_cast<unsigned long*>(dest)) = r;
732:   return true;
733: }
734:
735: bool Arg::parse_short_radix(const char* str,
736:                             int n,

```

```

737:         void* dest,
738:         int radix) {
739:     long r;
740:     if (!parse_long_radix(str, n, &r, radix)) return false; // Could not parse
741:     if (r < SHRT_MIN || r > SHRT_MAX) return false; // Out of range
742:     if (dest == NULL) return true;
743:     *(reinterpret_cast<short*>(dest)) = static_cast<short>(r);
744:     return true;
745: }
746:
747: bool Arg::parse_ushort_radix(const char* str,
748:         int n,
749:         void* dest,
750:         int radix) {
751:     unsigned long r;
752:     if (!parse_ulong_radix(str, n, &r, radix)) return false; // Could not parse
753:     if (r > USHRT_MAX) return false; // Out of range
754:     if (dest == NULL) return true;
755:     *(reinterpret_cast<unsigned short*>(dest)) = static_cast<unsigned short>(r);
756:     return true;
757: }
758:
759: bool Arg::parse_int_radix(const char* str,
760:         int n,
761:         void* dest,
762:         int radix) {
763:     long r;
764:     if (!parse_long_radix(str, n, &r, radix)) return false; // Could not parse
765:     if (r < INT_MIN || r > INT_MAX) return false; // Out of range
766:     if (dest == NULL) return true;
767:     *(reinterpret_cast<int*>(dest)) = r;
768:     return true;
769: }
770:
771: bool Arg::parse_uint_radix(const char* str,
772:         int n,
773:         void* dest,
774:         int radix) {
775:     unsigned long r;
776:     if (!parse_ulong_radix(str, n, &r, radix)) return false; // Could not parse
777:     if (r > UINT_MAX) return false; // Out of range

```

```

778: if (dest == NULL) return true;
779: *(reinterpret_cast<unsigned int*>(dest)) = r;
780: return true;
781: }
782:
783: bool Arg::parse_longlong_radix(const char* str,
784:                               int n,
785:                               void* dest,
786:                               int radix) {
787: #ifndef HAVE_LONG_LONG
788: return false;
789: #else
790: if (n == 0) return false;
791: char buf[kMaxNumberLength+1];
792: str = TerminateNumber(buf, str, n);
793: char* end;
794: errno = 0;
795: #if defined HAVE_STRTOQ
796: long long r = strtol(str, &end, radix);
797: //long long r = strtq(str, &end, radix);
798: #elif defined HAVE_STRTOLL
799: long long r = strtoll(str, &end, radix);
800: #elif defined HAVE__STRTOI64
801: long long r = _strtoi64(str, &end, radix);
802: #else
803: #error parse_longlong_radix: cannot convert input to a long-long
804: #endif
805: if (end != str + n) return false; // Leftover junk
806: if (errno) return false;
807: if (dest == NULL) return true;
808: *(reinterpret_cast<long long*>(dest)) = r;
809: return true;
810: #endif /* HAVE_LONG_LONG */
811: }
812:
813: bool Arg::parse_ulonglong_radix(const char* str,
814:                                 int n,
815:                                 void* dest,
816:                                 int radix) {
817: #ifndef HAVE_UNSIGNED_LONG_LONG
818: return false;

```

```

819: #else
820: if (n == 0) return false;
821: char buf[kMaxNumberLength+1];
822: str = TerminateNumber(buf, str, n);
823: if (str[0] == '-') return false; // strtoull() on a negative number?!
824: char* end;
825: errno = 0;
826: #if defined HAVE_STRTOQ
827: unsigned long r = strtoul(str, &end, radix);
828: //unsigned long long r = strtouq(str, &end, radix);
829: #elif defined HAVE_STRTOLL
830: unsigned long long r = strtoull(str, &end, radix);
831: #elif defined HAVE__STRTOI64
832: unsigned long long r = _strtoui64(str, &end, radix);
833: #else
834: #error parse_ulonglong_radix: cannot convert input to a long-long
835: #endif
836: if (end != str + n) return false; // Leftover junk
837: if (errno) return false;
838: if (dest == NULL) return true;
839: *(reinterpret_cast<unsigned long long*>(dest)) = r;
840: return true;
841: #endif /* HAVE_UNSIGNED_LONG_LONG */
842: }
843:
844: bool Arg::parse_double(const char* str, int n, void* dest) {
845: if (n == 0) return false;
846: static const int kMaxLength = 200;
847: char buf[kMaxLength];
848: if (n >= kMaxLength) return false;
849: memcpy(buf, str, n);
850: buf[n] = '\0';
851: errno = 0;
852: char* end;
853: double r = strtod(buf, &end);
854: if (end != buf + n) return false; // Leftover junk
855: if (errno) return false;
856: if (dest == NULL) return true;
857: *(reinterpret_cast<double*>(dest)) = r;
858: return true;
859: }

```

```

860:
861: bool Arg::parse_float(const char* str, int n, void* dest) {
862:     double r;
863:     if (!parse_double(str, n, &r)) return false;
864:     if (dest == NULL) return true;
865:     *(reinterpret_cast<float*>(dest)) = static_cast<float>(r);
866:     return true;
867: }
868:
869:
870: #define DEFINE_INTEGER_PARSERS(name) \
871:     bool Arg::parse_##name(const char* str, int n, void* dest) { \
872:         return parse_##name##_radix(str, n, dest, 10); \
873:     } \
874:     bool Arg::parse_##name##_hex(const char* str, int n, void* dest) { \
875:         return parse_##name##_radix(str, n, dest, 16); \
876:     } \
877:     bool Arg::parse_##name##_octal(const char* str, int n, void* dest) { \
878:         return parse_##name##_radix(str, n, dest, 8); \
879:     } \
880:     bool Arg::parse_##name##_cradix(const char* str, int n, void* dest) { \
881:         return parse_##name##_radix(str, n, dest, 0); \
882:     }
883:
884: DEFINE_INTEGER_PARSERS(short) /* */
885: DEFINE_INTEGER_PARSERS(ushort) /* */
886: DEFINE_INTEGER_PARSERS(int) /* Don't use semicolons after these */
887: DEFINE_INTEGER_PARSERS(uint) /* statements because they can cause */
888: DEFINE_INTEGER_PARSERS(long) /* compiler warnings if the checking */
889: DEFINE_INTEGER_PARSERS(ulong) /* level is turned up high enough. */
890: DEFINE_INTEGER_PARSERS(longlong) /* */
891: DEFINE_INTEGER_PARSERS(ulonglong) /* */
892:
893: #undef DEFINE_INTEGER_PARSERS
894:
895: } // namespace pcrecpp

```

File: sdm/VxWorks/libRegex/pcreposix.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3: *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module is a wrapper that provides a POSIX API to the underlying PCRE
42: functions. */
43:
44:
45: #ifdef HAVE_CONFIG_H
46: #include "config.h"
47: #endif
48:
49:
50: /* Ensure that the PCREPOSIX_EXP_xxx macros are set appropriately for
51: compiling these functions. This must come before including pcreposix.h, where
52: they are set for an application (using these functions) if they have not
53: previously been set. */
54:
55: #if defined(_WIN32) && !defined(PCRE_STATIC)
56: # define PCREPOSIX_EXP_DECL extern __declspec(dllexport)
57: # define PCREPOSIX_EXP_DEFN __declspec(dllexport)
58: #endif
59:
60: #include "pcre.h"
61: #include "pcre_internal.h"
62: #include "pcreposix.h"
63:
64:
65: /* Table to translate PCRE compile time error codes into POSIX error codes. */
66:
67: static const int eint[] = {
68: 0, /* no error */
69: REG_EESCAPE, /* \ at end of pattern */
70: REG_EESCAPE, /* \c at end of pattern */
71: REG_EESCAPE, /* unrecognized character follows \ */
72: REG_BADBR, /* numbers out of order in {} quantifier */
73: REG_BADBR, /* number too big in {} quantifier */
74: REG_EBRACK, /* missing terminating ] for character class */
75: REG_ECTYPE, /* invalid escape sequence in character class */
76: REG_ERANGE, /* range out of order in character class */

```

77: REG_BADRPT, /* nothing to repeat */
 78: REG_BADRPT, /* operand of unlimited repeat could match the empty string */
 79: REG_ASSERT, /* internal error: unexpected repeat */
 80: REG_BADPAT, /* unrecognized character after (? */
 81: REG_BADPAT, /* POSIX named classes are supported only within a class */
 82: REG_EPAREN, /* missing) */
 83: REG_ESUBREG, /* reference to non-existent subpattern */
 84: REG_INVARG, /* erroffset passed as NULL */
 85: REG_INVARG, /* unknown option bit(s) set */
 86: REG_EPAREN, /* missing) after comment */
 87: REG_ESIZE, /* parentheses nested too deeply */
 88: REG_ESIZE, /* regular expression too large */
 89: REG_ESPACE, /* failed to get memory */
 90: REG_EPAREN, /* unmatched brackets */
 91: REG_ASSERT, /* internal error: code overflow */
 92: REG_BADPAT, /* unrecognized character after (?< */
 93: REG_BADPAT, /* lookbehind assertion is not fixed length */
 94: REG_BADPAT, /* malformed number or name after (? (*/
 95: REG_BADPAT, /* conditional group contains more than two branches */
 96: REG_BADPAT, /* assertion expected after (? (*/
 97: REG_BADPAT, /* (?R or (?[+-]digits must be followed by) */
 98: REG_ETYPE, /* unknown POSIX class name */
 99: REG_BADPAT, /* POSIX collating elements are not supported */
 100: REG_INVARG, /* this version of PCRE is not compiled with PCRE_UTF8 support */
 101: REG_BADPAT, /* spare error */
 102: REG_BADPAT, /* character value in \x{...} sequence is too large */
 103: REG_BADPAT, /* invalid condition (? (0) */
 104: REG_BADPAT, /* \C not allowed in lookbehind assertion */
 105: REG_EESCAPE, /* PCRE does not support \L, \l, \N, \U, or \u */
 106: REG_BADPAT, /* number after (?C is > 255 */
 107: REG_BADPAT, /* closing) for (?C expected */
 108: REG_BADPAT, /* recursive call could loop indefinitely */
 109: REG_BADPAT, /* unrecognized character after (?P */
 110: REG_BADPAT, /* syntax error in subpattern name (missing terminator) */
 111: REG_BADPAT, /* two named subpatterns have the same name */
 112: REG_BADPAT, /* invalid UTF-8 string */
 113: REG_BADPAT, /* support for \P, \p, and \X has not been compiled */
 114: REG_BADPAT, /* malformed \P or \p sequence */
 115: REG_BADPAT, /* unknown property name after \P or \p */
 116: REG_BADPAT, /* subpattern name is too long (maximum 32 characters) */
 117: REG_BADPAT, /* too many named subpatterns (maximum 10,000) */


```

118: REG_BADPAT, /* repeated subpattern is too long */
119: REG_BADPAT, /* octal value is greater than \377 (not in UTF-8 mode) */
120: REG_BADPAT, /* internal error: overran compiling workspace */
121: REG_BADPAT, /* internal error: previously-checked referenced subpattern not found */
122: REG_BADPAT, /* DEFINE group contains more than one branch */
123: REG_BADPAT, /* repeating a DEFINE group is not allowed */
124: REG_INVARG, /* inconsistent NEWLINE options */
125: REG_BADPAT, /* \g is not followed followed by an (optionally braced) non-zero number */
126: REG_BADPAT, /* (?+ or (?- must be followed by a non-zero number */
127: REG_BADPAT, /* number is too big */
128: REG_BADPAT, /* subpattern name expected */
129: REG_BADPAT, /* digit expected after (?+ */
130: REG_BADPAT /* ] is an invalid data character in JavaScript compatibility mode */
131: };
132:
133: /* Table of texts corresponding to POSIX error codes */
134:
135: static const char *const pstring[] = {
136: "", /* Dummy for value 0 */
137: "internal error", /* REG_ASSERT */
138: "invalid repeat counts in {}", /* BADBR */
139: "pattern error", /* BADPAT */
140: "? * + invalid", /* BADRPT */
141: "unbalanced {}", /* EBRACE */
142: "unbalanced []", /* EBRACK */
143: "collation error - not relevant", /* ECOLLATE */
144: "bad class", /* ECTYPE */
145: "bad escape sequence", /* EESCAPE */
146: "empty expression", /* EMPTY */
147: "unbalanced ()", /* EPAREN */
148: "bad range inside []", /* ERANGE */
149: "expression too big", /* ESIZE */
150: "failed to get memory", /* ESPACE */
151: "bad back reference", /* ESUBREG */
152: "bad argument", /* INVARG */
153: "match failed" /* NOMATCH */
154: };
155:
156:
157:
158:

```

```

159: /*****
160: *      Translate error code to string      *
161: *****/
162:
163: PCREPOSIX_EXP_DEFN size_t PCRE_CALL_CONVENTION
164: regerror(int errcode, const regex_t *preg, char *errbuf, size_t errbuf_size)
165: {
166:     const char *message, *addmessage;
167:     size_t length, addlength;
168:
169:     message = (errcode >= (int)(sizeof(pstring)/sizeof(char *)))?
170:     "unknown error code" : pstring[errcode];
171:     length = strlen(message) + 1;
172:
173:     addmessage = " at offset ";
174:     addlength = (preg != NULL && (int)preg->re_erroffset != -1)?
175:     strlen(addmessage) + 6 : 0;
176:
177:     if (errbuf_size > 0)
178:     {
179:         if (addlength > 0 && errbuf_size >= length + addlength)
180:             sprintf(errbuf, "%s%s%-6d", message, addmessage, (int)preg->re_erroffset);
181:         else
182:         {
183:             strncpy(errbuf, message, errbuf_size - 1);
184:             errbuf[errbuf_size-1] = 0;
185:         }
186:     }
187:
188:     return length + addlength;
189: }
190:
191:
192:
193:
194: /*****
195: *      Free store held by a regex      *
196: *****/
197:
198: PCREPOSIX_EXP_DEFN void PCRE_CALL_CONVENTION
199: regfree(regex_t *preg)

```

```

200: {
201: (pcre_free)(preg->re_pcre);
202: }
203:
204:
205:
206:
207: /*****
208: *      Compile a regular expression      *
209: *****/
210:
211: /*
212: Arguments:
213: preg      points to a structure for recording the compiled expression
214: pattern    the pattern to compile
215: cflags     compilation flags
216:
217: Returns:   0 on success
218:            various non-zero codes on failure
219: */
220:
221: PCREPOSIX_EXP_DEFN int PCRE_CALL_CONVENTION
222: regcomp(regex_t *preg, const char *pattern, int cflags)
223: {
224: const char *errorptr;
225: int erroffset;
226: int errorcode;
227: int options = 0;
228:
229: if ((cflags & REG_ICASE) != 0) options |= PCRE_CASELESS;
230: if ((cflags & REG_NEWLINE) != 0) options |= PCRE_MULTILINE;
231: if ((cflags & REG_DOTALL) != 0) options |= PCRE_DOTALL;
232: if ((cflags & REG_NOSUB) != 0) options |= PCRE_NO_AUTO_CAPTURE;
233: if ((cflags & REG_UTF8) != 0) options |= PCRE_UTF8;
234:
235: preg->re_pcre = pcre_compile2(pattern, options, &errorcode, &errorptr,
236: &erroffset, NULL);
237: preg->re_erroffset = erroffset;
238:
239: if (preg->re_pcre == NULL) return eint[errorcode];
240:

```

```

241: preg->re_nsub = pcre_info((const pcre *)preg->re_pcre, NULL, NULL);
242: return 0;
243: }
244:
245:
246:
247:
248: /*****
249: *          Match a regular expression          *
250: *****/
251:
252: /* Unfortunately, PCRE requires 3 ints of working space for each captured
253: substring, so we have to get and release working store instead of just using
254: the POSIX structures as was done in earlier releases when PCRE needed only 2
255: ints. However, if the number of possible capturing brackets is small, use a
256: block of store on the stack, to reduce the use of malloc/free. The threshold is
257: in a macro that can be changed at configure time.
258:
259: If REG_NOSUB was specified at compile time, the PCRE_NO_AUTO_CAPTURE flag will
260: be set. When this is the case, the nmatch and pmatch arguments are ignored, and
261: the only result is yes/no/error. */
262:
263: PCREPOSIX_EXP_DEFN int PCRE_CALL_CONVENTION
264: regexexec(const regex_t *preg, const char *string, size_t nmatch,
265:  regmatch_t pmatch[], int eflags)
266: {
267: int rc, so, eo;
268: int options = 0;
269: int *ovector = NULL;
270: int small_ovector[POSIX_MALLOC_THRESHOLD * 3];
271: BOOL allocated_ovector = FALSE;
272: BOOL nosub =
273: (((const pcre *)preg->re_pcre)->options & PCRE_NO_AUTO_CAPTURE) != 0;
274:
275: if ((eflags & REG_NOTBOL) != 0) options |= PCRE_NOTBOL;
276: if ((eflags & REG_NOTEOL) != 0) options |= PCRE_NOTEOL;
277:
278: ((regex_t *)preg)->re_erroffset = (size_t)(-1); /* Only has meaning after compile */
279:
280: /* When no string data is being returned, ensure that nmatch is zero.
281: Otherwise, ensure the vector for holding the return data is large enough. */

```

```

282:
283: if (nosub) nmatch = 0;
284:
285: else if (nmatch > 0)
286: {
287: if (nmatch <= POSIX_MALLOC_THRESHOLD)
288: {
289:  ovector = &(small_ovector[0]);
290: }
291: else
292: {
293:  if (nmatch > INT_MAX/(sizeof(int) * 3)) return REG_ESPACE;
294:  ovector = (int *)malloc(sizeof(int) * nmatch * 3);
295:  if (ovector == NULL) return REG_ESPACE;
296:  allocated_ovector = TRUE;
297: }
298: }
299:
300: /* REG_STARTEND is a BSD extension, to allow for non-NUL-terminated strings.
301: The man page from OS X says "REG_STARTEND affects only the location of the
302: string, not how it is matched". That is why the "so" value is used to bump the
303: start location rather than being passed as a PCRE "starting offset". */
304:
305: if ((eflags & REG_STARTEND) != 0)
306: {
307:  so = pmatch[0].rm_so;
308:  eo = pmatch[0].rm_eo;
309: }
310: else
311: {
312:  so = 0;
313:  eo = strlen(string);
314: }
315:
316: rc = pcre_exec((const pcre *)preg->re_pcre, NULL, string + so, (eo - so),
317: 0, options, ovector, nmatch * 3);
318:
319: if (rc == 0) rc = nmatch; /* All captured slots were filled in */
320:
321: if (rc >= 0)
322: {

```

```

323: size_t i;
324: if (!nosub)
325: {
326:   for (i = 0; i < (size_t)rc; i++)
327:   {
328:     pmatch[i].rm_so = ovector[i*2];
329:     pmatch[i].rm_eo = ovector[i*2+1];
330:   }
331:   if (allocated_ovector) free(ovector);
332:   for (; i < nmatch; i++) pmatch[i].rm_so = pmatch[i].rm_eo = -1;
333: }
334: return 0;
335: }
336:
337: else
338: {
339:   if (allocated_ovector) free(ovector);
340:   switch(rc)
341:   {
342:     case PCRE_ERROR_NOMATCH: return REG_NOMATCH;
343:     case PCRE_ERROR_NULL: return REG_INVARG;
344:     case PCRE_ERROR_BADOPTION: return REG_INVARG;
345:     case PCRE_ERROR_BADMAGIC: return REG_INVARG;
346:     case PCRE_ERROR_UNKNOWN_NODE: return REG_ASSERT;
347:     case PCRE_ERROR_NOMEMORY: return REG_ESPACE;
348:     case PCRE_ERROR_MATCHLIMIT: return REG_ESPACE;
349:     case PCRE_ERROR_BADUTF8: return REG_INVARG;
350:     case PCRE_ERROR_BADUTF8_OFFSET: return REG_INVARG;
351:     default: return REG_ASSERT;
352:   }
353: }
354: }
355:
356: /* End of pcreposix.c */

```

File: sdm/VxWorks/libRegex/pcrecpp.h

```
1: // Copyright (c) 2005, Google Inc.
2: // All rights reserved.
3: //
4: // Redistribution and use in source and binary forms, with or without
5: // modification, are permitted provided that the following conditions are
6: // met:
7: //
8: //   * Redistributions of source code must retain the above copyright
9: // notice, this list of conditions and the following disclaimer.
10: //   * Redistributions in binary form must reproduce the above
11: // copyright notice, this list of conditions and the following disclaimer
12: // in the documentation and/or other materials provided with the
13: // distribution.
14: //   * Neither the name of Google Inc. nor the names of its
15: // contributors may be used to endorse or promote products derived from
16: // this software without specific prior written permission.
17: //
18: // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
19: // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
20: // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
21: // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
22: // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
23: // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
24: // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
25: // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
26: // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27: // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
28: // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
29: //
30: // Author: Sanjay Ghemawat
31: // Support for PCRE_XXX modifiers added by Giuseppe Maxia, July 2005
32:
33: #ifndef _PCRECPP_H
34: #define _PCRECPP_H
35:
36: // C++ interface to the pcre regular-expression library. RE supports
37: // Perl-style regular expressions (with extensions like \d, \w, \s,
38: // ...).
39: //
```

```

40: // -----
41: // REGEXP SYNTAX:
42: //
43: // This module is part of the pcre library and hence supports its syntax
44: // for regular expressions.
45: //
46: // The syntax is pretty similar to Perl's. For those not familiar
47: // with Perl's regular expressions, here are some examples of the most
48: // commonly used extensions:
49: //
50: // "hello (\w+) world" -- \w matches a "word" character
51: // "version (\d+)"    -- \d matches a digit
52: // "hello \s+world"   -- \s matches any whitespace character
53: // "\b(\w+)\b"        -- \b matches empty string at a word boundary
54: // "(?i)hello"        -- (?i) turns on case-insensitive matching
55: // "/\s*(.*)\s*/"     -- .*? matches . minimum no. of times possible
56: //
57: // -----
58: // MATCHING INTERFACE:
59: //
60: // The "FullMatch" operation checks that supplied text matches a
61: // supplied pattern exactly.
62: //
63: // Example: successful match
64: // pcrecpp::RE re("h.*o");
65: // re.FullMatch("hello");
66: //
67: // Example: unsuccessful match (requires full match):
68: // pcrecpp::RE re("e");
69: // !re.FullMatch("hello");
70: //
71: // Example: creating a temporary RE object:
72: // pcrecpp::RE("h.*o").FullMatch("hello");
73: //
74: // You can pass in a "const char*" or a "string" for "text". The
75: // examples below tend to use a const char*.
76: //
77: // You can, as in the different examples above, store the RE object
78: // explicitly in a variable or use a temporary RE object. The
79: // examples below use one mode or the other arbitrarily. Either
80: // could correctly be used for any of these examples.

```



```

81: //
82: // -----
83: // MATCHING WITH SUB-STRING EXTRACTION:
84: //
85: // You can supply extra pointer arguments to extract matched subpieces.
86: //
87: // Example: extracts "ruby" into "s" and 1234 into "i"
88: //   int i;
89: //   string s;
90: //   pcrecpp::RE re("(\\w+):(\\d+)");
91: //   re.FullMatch("ruby:1234", &s, &i);
92: //
93: // Example: does not try to extract any extra sub-patterns
94: //   re.FullMatch("ruby:1234", &s);
95: //
96: // Example: does not try to extract into NULL
97: //   re.FullMatch("ruby:1234", NULL, &i);
98: //
99: // Example: integer overflow causes failure
100: //   !re.FullMatch("ruby:1234567891234", NULL, &i);
101: //
102: // Example: fails because there aren't enough sub-patterns:
103: //   !pcrecpp::RE("(\\w+:\\d+").FullMatch("ruby:1234", &s);
104: //
105: // Example: fails because string cannot be stored in integer
106: //   !pcrecpp::RE("(.*").FullMatch("ruby", &i);
107: //
108: // The provided pointer arguments can be pointers to any scalar numeric
109: // type, or one of
110: //   string      (matched piece is copied to string)
111: //   StringPiece (StringPiece is mutated to point to matched piece)
112: //   T           (where "bool T::ParseFrom(const char*, int)" exists)
113: //   NULL        (the corresponding matched sub-pattern is not copied)
114: //
115: // CAVEAT: An optional sub-pattern that does not exist in the matched
116: // string is assigned the empty string. Therefore, the following will
117: // return false (because the empty string is not a valid number):
118: //   int number;
119: //   pcrecpp::RE::FullMatch("abc", "[a-z]+(\\d+)?", &number);
120: //
121: // -----

```

```

122: // DO_MATCH
123: //
124: // The matching interface supports at most 16 arguments per call.
125: // If you need more, consider using the more general interface
126: // pcrecpp::RE::DoMatch(). See pcrecpp.h for the signature for DoMatch.
127: //
128: // -----
129: // PARTIAL MATCHES
130: //
131: // You can use the "PartialMatch" operation when you want the pattern
132: // to match any substring of the text.
133: //
134: // Example: simple search for a string:
135: //   pcrecpp::RE("ell").PartialMatch("hello");
136: //
137: // Example: find first number in a string:
138: //   int number;
139: //   pcrecpp::RE re("(\\d+)");
140: //   re.PartialMatch("x*100 + 20", &number);
141: //   assert(number == 100);
142: //
143: // -----
144: // UTF-8 AND THE MATCHING INTERFACE:
145: //
146: // By default, pattern and text are plain text, one byte per character.
147: // The UTF8 flag, passed to the constructor, causes both pattern
148: // and string to be treated as UTF-8 text, still a byte stream but
149: // potentially multiple bytes per character. In practice, the text
150: // is likelier to be UTF-8 than the pattern, but the match returned
151: // may depend on the UTF8 flag, so always use it when matching
152: // UTF8 text. E.g., "." will match one byte normally but with UTF8
153: // set may match up to three bytes of a multi-byte character.
154: //
155: // Example:
156: //   pcrecpp::RE_Options options;
157: //   options.set_utf8();
158: //   pcrecpp::RE re(utf8_pattern, options);
159: //   re.FullMatch(utf8_string);
160: //
161: // Example: using the convenience function UTF8():
162: //   pcrecpp::RE re(utf8_pattern, pcrecpp::UTF8());

```

```

163: // re.FullMatch(utf8_string);
164: //
165: // NOTE: The UTF8 option is ignored if pcre was not configured with the
166: // --enable-utf8 flag.
167: //
168: // -----
169: // PASSING MODIFIERS TO THE REGULAR EXPRESSION ENGINE
170: //
171: // PCRE defines some modifiers to change the behavior of the regular
172: // expression engine.
173: // The C++ wrapper defines an auxiliary class, RE_Options, as a vehicle
174: // to pass such modifiers to a RE class.
175: //
176: // Currently, the following modifiers are supported
177: //
178: // modifier      description      Perl corresponding
179: //
180: // PCRE_CASELESS  case insensitive match  /i
181: // PCRE_MULTILINE multiple lines match    /m
182: // PCRE_DOTALL    dot matches newlines    /s
183: // PCRE_DOLLAR_ENDONLY $ matches only at end  N/A
184: // PCRE_EXTRA     strict escape parsing   N/A
185: // PCRE_EXTENDED  ignore whitespaces     /x
186: // PCRE_UTF8      handles UTF8 chars     built-in
187: // PCRE_UNGREEDY  reverses * and *?      N/A
188: // PCRE_NO_AUTO_CAPTURE disables matching parens N/A (*)
189: //
190: // (For a full account on how each modifier works, please check the
191: // PCRE API reference manual).
192: //
193: // (*) Both Perl and PCRE allow non matching parentheses by means of the
194: // "?" modifier within the pattern itself. e.g. (?:ab|cd) does not
195: // capture, while (ab|cd) does.
196: //
197: // For each modifier, there are two member functions whose name is made
198: // out of the modifier in lowercase, without the "PCRE_" prefix. For
199: // instance, PCRE_CASELESS is handled by
200: // bool caseless(),
201: // which returns true if the modifier is set, and
202: // RE_Options & set_caseless(bool),
203: // which sets or unsets the modifier.

```

```

204: //
205: // Moreover, PCRE_EXTRA_MATCH_LIMIT can be accessed through the
206: // set_match_limit() and match_limit() member functions.
207: // Setting match_limit to a non-zero value will limit the execution of
208: // pcre to keep it from doing bad things like blowing the stack or taking
209: // an eternity to return a result. A value of 5000 is good enough to stop
210: // stack blowup in a 2MB thread stack. Setting match_limit to zero will
211: // disable match limiting. Alternately, you can set match_limit_recursion()
212: // which uses PCRE_EXTRA_MATCH_LIMIT_RECURSION to limit how much pcre
213: // recurses. match_limit() caps the number of matches pcre does;
214: // match_limit_recursion() caps the depth of recursion.
215: //
216: // Normally, to pass one or more modifiers to a RE class, you declare
217: // a RE_Options object, set the appropriate options, and pass this
218: // object to a RE constructor. Example:
219: //
220: //   RE_options opt;
221: //   opt.set_caseless(true);
222: //
223: //   if (RE("HELLO", opt).PartialMatch("hello world")) ...
224: //
225: // RE_options has two constructors. The default constructor takes no
226: // arguments and creates a set of flags that are off by default.
227: //
228: // The optional parameter 'option_flags' is to facilitate transfer
229: // of legacy code from C programs. This lets you do
230: //   RE(pattern, RE_Options(PCRE_CASELESS|PCRE_MULTILINE)).PartialMatch(str);
231: //
232: // But new code is better off doing
233: //   RE(pattern,
234: //       RE_Options().set_caseless(true).set_multiline(true)).PartialMatch(str);
235: // (See below)
236: //
237: // If you are going to pass one of the most used modifiers, there are some
238: // convenience functions that return a RE_Options class with the
239: // appropriate modifier already set:
240: // CASELESS(), UTF8(), MULTILINE(), DOTALL(), EXTENDED()
241: //
242: // If you need to set several options at once, and you don't want to go
243: // through the pains of declaring a RE_Options object and setting several
244: // options, there is a parallel method that give you such ability on the

```

```

245: // fly. You can concatenate several set_XXXXXX member functions, since each
246: // of them returns a reference to its class object. e.g.: to pass
247: // PCRE_CASELESS, PCRE_EXTENDED, and PCRE_MULTILINE to a RE with one
248: // statement, you may write
249: //
250: // RE(" ^ xyz \s+ .* blah$", RE_Options()
251: //         .set_caseless(true)
252: //         .set_extended(true)
253: //         .set_multiline(true)).PartialMatch(sometext);
254: //
255: // -----
256: // SCANNING TEXT INCREMENTALLY
257: //
258: // The "Consume" operation may be useful if you want to repeatedly
259: // match regular expressions at the front of a string and skip over
260: // them as they match. This requires use of the "StringPiece" type,
261: // which represents a sub-range of a real string. Like RE, StringPiece
262: // is defined in the pcrecpp namespace.
263: //
264: // Example: read lines of the form "var = value" from a string.
265: // string contents = ...;           // Fill string somehow
266: // pcrecpp::StringPiece input(contents); // Wrap in a StringPiece
267: //
268: // string var;
269: // int value;
270: // pcrecpp::RE re("(\\w+) = (\\d+)\\n");
271: // while (re.Consume(&input, &var, &value)) {
272: //     ...;
273: // }
274: //
275: // Each successful call to "Consume" will set "var/value", and also
276: // advance "input" so it points past the matched text.
277: //
278: // The "FindAndConsume" operation is similar to "Consume" but does not
279: // anchor your match at the beginning of the string. For example, you
280: // could extract all words from a string by repeatedly calling
281: // pcrecpp::RE("(\\w+)").FindAndConsume(&input, &word)
282: //
283: // -----
284: // PARSING HEX/OCTAL/C-RADIX NUMBERS
285: //

```

```

286: // By default, if you pass a pointer to a numeric value, the
287: // corresponding text is interpreted as a base-10 number. You can
288: // instead wrap the pointer with a call to one of the operators Hex(),
289: // Octal(), or CRadix() to interpret the text in another base. The
290: // CRadix operator interprets C-style "0" (base-8) and "0x" (base-16)
291: // prefixes, but defaults to base-10.
292: //
293: // Example:
294: // int a, b, c, d;
295: // pcrecpp::RE re("(.*)(.*)"(.*)");
296: // re.FullMatch("100 40 0100 0x40",
297: //             pcrecpp::Octal(&a), pcrecpp::Hex(&b),
298: //             pcrecpp::CRadix(&c), pcrecpp::CRadix(&d));
299: // will leave 64 in a, b, c, and d.
300: //
301: // -----
302: // REPLACING PARTS OF STRINGS
303: //
304: // You can replace the first match of "pattern" in "str" with
305: // "rewrite". Within "rewrite", backslash-escaped digits ( \1 to \9)
306: // can be used to insert text matching corresponding parenthesized
307: // group from the pattern. \0 in "rewrite" refers to the entire
308: // matching text. E.g.,
309: //
310: // string s = "yabba dabba doo";
311: // pcrecpp::RE("b+").Replace("d", &s);
312: //
313: // will leave "s" containing "yada dabba doo". The result is true if
314: // the pattern matches and a replacement occurs, or false otherwise.
315: //
316: // GlobalReplace() is like Replace(), except that it replaces all
317: // occurrences of the pattern in the string with the rewrite.
318: // Replacements are not subject to re-matching. E.g.,
319: //
320: // string s = "yabba dabba doo";
321: // pcrecpp::RE("b+").GlobalReplace("d", &s);
322: //
323: // will leave "s" containing "yada dada doo". It returns the number
324: // of replacements made.
325: //
326: // Extract() is like Replace(), except that if the pattern matches,

```

```

327: // "rewrite" is copied into "out" (an additional argument) with
328: // substitutions. The non-matching portions of "text" are ignored.
329: // Returns true iff a match occurred and the extraction happened
330: // successfully. If no match occurs, the string is left unaffected.
331:
332:
333: #include <string>
334: #include <pcre.h>
335: #include <pcrecpparg.h> // defines the Arg class
336: // This isn't technically needed here, but we include it
337: // anyway so folks who include pcrecpp.h don't have to.
338: #include <pcre_stringpiece.h>
339:
340: namespace pcrecpp {
341:
342: #define PCRE_SET_OR_CLEAR(b, o) \
343:   if (b) all_options_ |= (o); else all_options_ &= ~(o); \
344:   return *this
345:
346: #define PCRE_IS_SET(o) \
347:   (all_options_ & o) == o
348:
349: /***** Compiling regular expressions: the RE class *****/
350:
351: // RE_Options allow you to set options to be passed along to pcre,
352: // along with other options we put on top of pcre.
353: // Only 9 modifiers, plus match_limit and match_limit_recursion,
354: // are supported now.
355: class PCRECPP_EXP_DEFN RE_Options {
356: public:
357:   // constructor
358:   RE_Options() : match_limit_(0), match_limit_recursion_(0), all_options_(0) {}
359:
360:   // alternative constructor.
361:   // To facilitate transfer of legacy code from C programs
362:   //
363:   // This lets you do
364:   //   RE(pattern, RE_Options(PCRE_CASELESS|PCRE_MULTILINE)).PartialMatch(str);
365:   // But new code is better off doing
366:   //   RE(pattern,
367:   //     RE_Options().set_caseless(true).set_multiline(true)).PartialMatch(str);

```

```

368: RE_Options(int option_flags) : match_limit_(0), match_limit_recursion_(0),
369:     all_options_(option_flags) {}
370: // we're fine with the default destructor, copy constructor, etc.
371:
372: // accessors and mutators
373: int match_limit() const { return match_limit_; };
374: RE_Options &set_match_limit(int limit) {
375:     match_limit_ = limit;
376:     return *this;
377: }
378:
379: int match_limit_recursion() const { return match_limit_recursion_; };
380: RE_Options &set_match_limit_recursion(int limit) {
381:     match_limit_recursion_ = limit;
382:     return *this;
383: }
384:
385: bool caseless() const {
386:     return PCRE_IS_SET(PCRE_CASELESS);
387: }
388: RE_Options &set_caseless(bool x) {
389:     PCRE_SET_OR_CLEAR(x, PCRE_CASELESS);
390: }
391:
392: bool multiline() const {
393:     return PCRE_IS_SET(PCRE_MULTILINE);
394: }
395: RE_Options &set_multiline(bool x) {
396:     PCRE_SET_OR_CLEAR(x, PCRE_MULTILINE);
397: }
398:
399: bool dotall() const {
400:     return PCRE_IS_SET(PCRE_DOTALL);
401: }
402: RE_Options &set_dotall(bool x) {
403:     PCRE_SET_OR_CLEAR(x, PCRE_DOTALL);
404: }
405:
406: bool extended() const {
407:     return PCRE_IS_SET(PCRE_EXTENDED);
408: }

```



```

409: RE_Options &set_extended(bool x) {
410:     PCRE_SET_OR_CLEAR(x, PCRE_EXTENDED);
411: }
412:
413: bool dollar_endonly() const {
414:     return PCRE_IS_SET(PCRE_DOLLAR_ENDONLY);
415: }
416: RE_Options &set_dollar_endonly(bool x) {
417:     PCRE_SET_OR_CLEAR(x, PCRE_DOLLAR_ENDONLY);
418: }
419:
420: bool extra() const {
421:     return PCRE_IS_SET(PCRE_EXTRA);
422: }
423: RE_Options &set_extra(bool x) {
424:     PCRE_SET_OR_CLEAR(x, PCRE_EXTRA);
425: }
426:
427: bool ungreedy() const {
428:     return PCRE_IS_SET(PCRE_UNGREEDY);
429: }
430: RE_Options &set_ungreedy(bool x) {
431:     PCRE_SET_OR_CLEAR(x, PCRE_UNGREEDY);
432: }
433:
434: bool utf8() const {
435:     return PCRE_IS_SET(PCRE_UTF8);
436: }
437: RE_Options &set_utf8(bool x) {
438:     PCRE_SET_OR_CLEAR(x, PCRE_UTF8);
439: }
440:
441: bool no_auto_capture() const {
442:     return PCRE_IS_SET(PCRE_NO_AUTO_CAPTURE);
443: }
444: RE_Options &set_no_auto_capture(bool x) {
445:     PCRE_SET_OR_CLEAR(x, PCRE_NO_AUTO_CAPTURE);
446: }
447:
448: RE_Options &set_all_options(int opt) {
449:     all_options_ = opt;

```

```

450:   return *this;
451: }
452: int all_options() const {
453:   return all_options_ ;
454: }
455:
456: // TODO: add other pcre flags
457:
458: private:
459:   int match_limit_;
460:   int match_limit_recursion_;
461:   int all_options_;
462: };
463:
464: // These functions return some common RE_Options
465: static inline RE_Options UTF8() {
466:   return RE_Options().set_utf8(true);
467: }
468:
469: static inline RE_Options CASELESS() {
470:   return RE_Options().set_caseless(true);
471: }
472: static inline RE_Options MULTILINE() {
473:   return RE_Options().set_multiline(true);
474: }
475:
476: static inline RE_Options DOTALL() {
477:   return RE_Options().set_dotall(true);
478: }
479:
480: static inline RE_Options EXTENDED() {
481:   return RE_Options().set_extended(true);
482: }
483:
484: // Interface for regular expression matching. Also corresponds to a
485: // pre-compiled regular expression. An "RE" object is safe for
486: // concurrent use by multiple threads.
487: class PCRECPP_EXP_DEFN RE {
488: public:
489:   // We provide implicit conversions from strings so that users can
490:   // pass in a string or a "const char*" wherever an "RE" is expected.

```

```

491: RE(const string& pat) { Init(pat, NULL); }
492: RE(const string& pat, const RE_Options& option) { Init(pat, &option); }
493: RE(const char* pat) { Init(pat, NULL); }
494: RE(const char* pat, const RE_Options& option) { Init(pat, &option); }
495: RE(const unsigned char* pat) {
496:     Init(reinterpret_cast<const char*>(pat), NULL);
497: }
498: RE(const unsigned char* pat, const RE_Options& option) {
499:     Init(reinterpret_cast<const char*>(pat), &option);
500: }
501:
502: // Copy constructor & assignment - note that these are expensive
503: // because they recompile the expression.
504: RE(const RE& re) { Init(re.pattern_, &re.options_); }
505: const RE& operator=(const RE& re) {
506:     if (this != &re) {
507:         Cleanup();
508:
509:         // This is the code that originally came from Google
510:         // Init(re.pattern_.c_str(), &re.options_);
511:
512:         // This is the replacement from Ari Pollak
513:         Init(re.pattern_, &re.options_);
514:     }
515:     return *this;
516: }
517:
518:
519: ~RE();
520:
521: // The string specification for this RE. E.g.
522: // RE re("ab*c?d+");
523: // re.pattern(); // "ab*c?d+"
524: const string& pattern() const { return pattern_; }
525:
526: // If RE could not be created properly, returns an error string.
527: // Else returns the empty string.
528: const string& error() const { return *error_; }
529:
530: /***** The useful part: the matching interface *****/
531:

```

```

532: // This is provided so one can do pattern.ReplaceAll() just as
533: // easily as ReplaceAll(pattern-text, ....)
534:
535: bool FullMatch(const StringPiece& text,
536:               const Arg& ptr1 = no_arg,
537:               const Arg& ptr2 = no_arg,
538:               const Arg& ptr3 = no_arg,
539:               const Arg& ptr4 = no_arg,
540:               const Arg& ptr5 = no_arg,
541:               const Arg& ptr6 = no_arg,
542:               const Arg& ptr7 = no_arg,
543:               const Arg& ptr8 = no_arg,
544:               const Arg& ptr9 = no_arg,
545:               const Arg& ptr10 = no_arg,
546:               const Arg& ptr11 = no_arg,
547:               const Arg& ptr12 = no_arg,
548:               const Arg& ptr13 = no_arg,
549:               const Arg& ptr14 = no_arg,
550:               const Arg& ptr15 = no_arg,
551:               const Arg& ptr16 = no_arg) const;
552:
553: bool PartialMatch(const StringPiece& text,
554:                  const Arg& ptr1 = no_arg,
555:                  const Arg& ptr2 = no_arg,
556:                  const Arg& ptr3 = no_arg,
557:                  const Arg& ptr4 = no_arg,
558:                  const Arg& ptr5 = no_arg,
559:                  const Arg& ptr6 = no_arg,
560:                  const Arg& ptr7 = no_arg,
561:                  const Arg& ptr8 = no_arg,
562:                  const Arg& ptr9 = no_arg,
563:                  const Arg& ptr10 = no_arg,
564:                  const Arg& ptr11 = no_arg,
565:                  const Arg& ptr12 = no_arg,
566:                  const Arg& ptr13 = no_arg,
567:                  const Arg& ptr14 = no_arg,
568:                  const Arg& ptr15 = no_arg,
569:                  const Arg& ptr16 = no_arg) const;
570:
571: bool Consume(StringPiece* input,
572:              const Arg& ptr1 = no_arg,

```

```

573:         const Arg& ptr2 = no_arg,
574:         const Arg& ptr3 = no_arg,
575:         const Arg& ptr4 = no_arg,
576:         const Arg& ptr5 = no_arg,
577:         const Arg& ptr6 = no_arg,
578:         const Arg& ptr7 = no_arg,
579:         const Arg& ptr8 = no_arg,
580:         const Arg& ptr9 = no_arg,
581:         const Arg& ptr10 = no_arg,
582:         const Arg& ptr11 = no_arg,
583:         const Arg& ptr12 = no_arg,
584:         const Arg& ptr13 = no_arg,
585:         const Arg& ptr14 = no_arg,
586:         const Arg& ptr15 = no_arg,
587:         const Arg& ptr16 = no_arg) const;
588:
589: bool FindAndConsume(StringPiece* input,
590:         const Arg& ptr1 = no_arg,
591:         const Arg& ptr2 = no_arg,
592:         const Arg& ptr3 = no_arg,
593:         const Arg& ptr4 = no_arg,
594:         const Arg& ptr5 = no_arg,
595:         const Arg& ptr6 = no_arg,
596:         const Arg& ptr7 = no_arg,
597:         const Arg& ptr8 = no_arg,
598:         const Arg& ptr9 = no_arg,
599:         const Arg& ptr10 = no_arg,
600:         const Arg& ptr11 = no_arg,
601:         const Arg& ptr12 = no_arg,
602:         const Arg& ptr13 = no_arg,
603:         const Arg& ptr14 = no_arg,
604:         const Arg& ptr15 = no_arg,
605:         const Arg& ptr16 = no_arg) const;
606:
607: bool Replace(const StringPiece& rewrite,
608:         string *str) const;
609:
610: int GlobalReplace(const StringPiece& rewrite,
611:         string *str) const;
612:
613: bool Extract(const StringPiece &rewrite,

```

```

614:         const StringPiece &text,
615:         string *out) const;
616:
617: // Escapes all potentially meaningful regexp characters in
618: // 'unquoted'. The returned string, used as a regular expression,
619: // will exactly match the original string. For example,
620: //      1.5-2.0?
621: // may become:
622: //      1\\.5\\-2\\.0\\?
623: // Note QuoteMeta behaves the same as perl's QuoteMeta function,
624: // *except* that it escapes the NUL character ( \0) as backslash + 0,
625: // rather than backslash + NUL.
626: static string QuoteMeta(const StringPiece& unquoted);
627:
628:
629: /***** Generic matching interface *****/
630:
631: // Type of match (TODO: Should be restructured as part of RE_Options)
632: enum Anchor {
633:     UNANCHORED,      // No anchoring
634:     ANCHOR_START,    // Anchor at start only
635:     ANCHOR_BOTH      // Anchor at start and end
636: };
637:
638: // General matching routine. Stores the length of the match in
639: // "*consumed" if successful.
640: bool DoMatch(const StringPiece& text,
641:             Anchor anchor,
642:             int* consumed,
643:             const Arg* const* args, int n) const;
644:
645: // Return the number of capturing subpatterns, or -1 if the
646: // regexp wasn't valid on construction.
647: int NumberOfCapturingGroups() const;
648:
649: // The default value for an argument, to indicate no arg was passed in
650: static Arg no_arg;
651:
652: private:
653:
654: void Init(const string& pattern, const RE_Options* options);

```

```

655: void Cleanup();
656:
657: // Match against "text", filling in "vec" (up to "vecsize" * 2/3) with
658: // pairs of integers for the beginning and end positions of matched
659: // text. The first pair corresponds to the entire matched text;
660: // subsequent pairs correspond, in order, to parentheses-captured
661: // matches. Returns the number of pairs (one more than the number of
662: // the last subpattern with a match) if matching was successful
663: // and zero if the match failed.
664: // I.e. for RE("(foo)|(bar)|(baz)") it will return 2, 3, and 4 when matching
665: // against "foo", "bar", and "baz" respectively.
666: // When matching RE("(foo)|hello") against "hello", it will return 1.
667: // But the values for all subpattern are filled in into "vec".
668: int TryMatch(const StringPiece& text,
669:             int startpos,
670:             Anchor anchor,
671:             int *vec,
672:             int vecsize) const;
673:
674: // Append the "rewrite" string, with backslash substitutions from "text"
675: // and "vec", to string "out".
676: bool Rewrite(string *out,
677:             const StringPiece& rewrite,
678:             const StringPiece& text,
679:             int *vec,
680:             int veclen) const;
681:
682: // internal implementation for DoMatch
683: bool DoMatchImpl(const StringPiece& text,
684:                 Anchor anchor,
685:                 int* consumed,
686:                 const Arg* const args[],
687:                 int n,
688:                 int* vec,
689:                 int vecsize) const;
690:
691: // Compile the regexp for the specified anchoring mode
692: pcre* Compile(Anchor anchor);
693:
694: string    pattern_;
695: RE_Options options_;

```

```
696: pcre*      re_full_;    // For full matches
697: pcre*      re_partial_;  // For partial matches
698: const string* error_;     // Error indicator (or points to empty string)
699: };
700:
701: } // namespace pcrecpp
702:
703: #endif /* _PCRECPP_H */
```


File: sdm/VxWorks/libRegex/pcre_exec.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
38: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
39: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains pcre_exec(), the externally visible function that does
42: pattern matching using an NFA algorithm, trying to mimic Perl as closely as
43: possible. There are also some static supporting functions. */
44:
45: #ifdef HAVE_CONFIG_H
46: #include "config.h"
47: #endif
48:
49: #define NLBLOCK md          /* Block containing newline information */
50: #define PSSTART start_subject /* Field containing processed string start */
51: #define PSEND end_subject   /* Field containing processed string end */
52:
53: #include "pcre_internal.h"
54:
55: /* Undefine some potentially clashing cpp symbols */
56:
57: #undef min
58: #undef max
59:
60: /* Flag bits for the match() function */
61:
62: #define match_condassert 0x01 /* Called to check a condition assertion */
63: #define match_cbegroup 0x02 /* Could-be-empty unlimited repeat group */
64:
65: /* Non-error returns from the match() function. Error returns are externally
66: defined PCRE_ERROR_xxx codes, which are all negative. */
67:
68: #define MATCH_MATCH 1
69: #define MATCH_NOMATCH 0
70:
71: /* Special internal returns from the match() function. Make them sufficiently
72: negative to avoid the external error codes. */
73:
74: #define MATCH_COMMIT (-999)
75: #define MATCH_PRUNE (-998)
76: #define MATCH_SKIP (-997)

```

```

77: #define MATCH_THEN      (-996)
78:
79: /* Maximum number of ints of offset to save on the stack for recursive calls.
80: If the offset vector is bigger, malloc is used. This should be a multiple of 3,
81: because the offset vector is always a multiple of 3 long. */
82:
83: #define REC_STACK_SAVE_MAX 30
84:
85: /* Min and max values for the common repeats; for the maxima, 0 => infinity */
86:
87: static const char rep_min[] = { 0, 0, 1, 1, 0, 0 };
88: static const char rep_max[] = { 0, 0, 0, 0, 1, 1 };
89:
90:
91:
92: #ifdef DEBUG
93: /*****
94:  *      Debugging function to print chars      *
95:  *****/
96:
97: /* Print a sequence of chars in printable format, stopping at the end of the
98: subject if the requested.
99:
100: Arguments:
101:  p      points to characters
102:  length  number to print
103:  is_subject TRUE if printing from within md->start_subject
104:  md      pointer to matching data block, if is_subject is TRUE
105:
106: Returns:  nothing
107: */
108:
109: static void
110: pchars(const uschar *p, int length, BOOL is_subject, match_data *md)
111: {
112: unsigned int c;
113: if (is_subject && length > md->end_subject - p) length = md->end_subject - p;
114: while (length-- > 0)
115:   if (isprint(c = *(p++))) printf("%c", c); else printf(" \\x%02x", c);
116: }
117: #endif

```

```

118:
119:
120:
121: /*****
122: *      Match a back-reference      *
123: *****/
124:
125: /* If a back reference hasn't been set, the length that is passed is greater
126: than the number of characters left in the string, so the match fails.
127:
128: Arguments:
129: offset    index into the offset vector
130: eptr      points into the subject
131: length    length to be matched
132: md        points to match data block
133: ims       the ims flags
134:
135: Returns:   TRUE if matched
136: */
137:
138: static BOOL
139: match_ref(int offset, register USPTR eptr, int length, match_data *md,
140: unsigned long int ims)
141: {
142: USPTR p = md->start_subject + md->offset_vector[offset];
143:
144: #ifdef DEBUG
145: if (eptr >= md->end_subject)
146: printf("matching subject <null>");
147: else
148: {
149: printf("matching subject ");
150: pchars(eptr, length, TRUE, md);
151: }
152: printf(" against backref ");
153: pchars(p, length, FALSE, md);
154: printf("\n");
155: #endif
156:
157: /* Always fail if not enough characters left */
158:

```

```

159: if (length > md->end_subject - eptr) return FALSE;
160:
161: /* Separate the caseless case for speed. In UTF-8 mode we can only do this
162: properly if Unicode properties are supported. Otherwise, we can check only
163: ASCII characters. */
164:
165: if ((ims & PCRE_CASELESS) != 0)
166: {
167: #ifdef SUPPORT_UTF8
168: #ifdef SUPPORT_UCP
169:   if (md->utf8)
170:   {
171:     USPTR endptr = eptr + length;
172:     while (eptr < endptr)
173:     {
174:       int c, d;
175:       GETCHARINC(c, eptr);
176:       GETCHARINC(d, p);
177:       if (c != d && c != UCD_OTHERCASE(d)) return FALSE;
178:     }
179:   }
180: else
181: #endif
182: #endif
183:
184: /* The same code works when not in UTF-8 mode and in UTF-8 mode when there
185: is no UCP support. */
186:
187: while (length-- > 0)
188:   { if (md->lcc[*p++] != md->lcc[*eptr++]) return FALSE; }
189: }
190:
191: /* In the caseful case, we can just compare the bytes, whether or not we
192: are in UTF-8 mode. */
193:
194: else
195:   { while (length-- > 0) if (*p++ != *eptr++) return FALSE; }
196:
197: return TRUE;
198: }
199:

```

```

200:
201:
202: /*****
203: *****/
204:         RECURSION IN THE match() FUNCTION
205:
206: The match() function is highly recursive, though not every recursive call
207: increases the recursive depth. Nevertheless, some regular expressions can cause
208: it to recurse to a great depth. I was writing for Unix, so I just let it call
209: itself recursively. This uses the stack for saving everything that has to be
210: saved for a recursive call. On Unix, the stack can be large, and this works
211: fine.
212:
213: It turns out that on some non-Unix-like systems there are problems with
214: programs that use a lot of stack. (This despite the fact that every last chip
215: has oodles of memory these days, and techniques for extending the stack have
216: been known for decades.) So...
217:
218: There is a fudge, triggered by defining NO_RECURSE, which avoids recursive
219: calls by keeping local variables that need to be preserved in blocks of memory
220: obtained from malloc() instead instead of on the stack. Macros are used to
221: achieve this so that the actual code doesn't look very different to what it
222: always used to.
223:
224: The original heap-recursive code used longjmp(). However, it seems that this
225: can be very slow on some operating systems. Following a suggestion from Stan
226: Switzer, the use of longjmp() has been abolished, at the cost of having to
227: provide a unique number for each call to RMATCH. There is no way of generating
228: a sequence of numbers at compile time in C. I have given them names, to make
229: them stand out more clearly.
230:
231: Crude tests on x86 Linux show a small speedup of around 5-8%. However, on
232: FreeBSD, avoiding longjmp() more than halves the time taken to run the standard
233: tests. Furthermore, not using longjmp() means that local dynamic variables
234: don't have indeterminate values; this has meant that the frame size can be
235: reduced because the result can be "passed back" by straight setting of the
236: variable instead of being passed in the frame.
237: *****/
238: *****/
239:
240: /* Numbers for RMATCH calls. When this list is changed, the code at HEAP_RETURN

```

```

241: below must be updated in sync. */
242:
243: enum { RM1=1, RM2, RM3, RM4, RM5, RM6, RM7, RM8, RM9, RM10,
244:      RM11, RM12, RM13, RM14, RM15, RM16, RM17, RM18, RM19, RM20,
245:      RM21, RM22, RM23, RM24, RM25, RM26, RM27, RM28, RM29, RM30,
246:      RM31, RM32, RM33, RM34, RM35, RM36, RM37, RM38, RM39, RM40,
247:      RM41, RM42, RM43, RM44, RM45, RM46, RM47, RM48, RM49, RM50,
248:      RM51, RM52, RM53, RM54 };
249:
250: /* These versions of the macros use the stack, as normal. There are debugging
251: versions and production versions. Note that the "rw" argument of RMATCH isn't
252: actuall used in this definition. */
253:
254: #ifndef NO_RECURSE
255: #define REGISTER register
256:
257: #ifdef DEBUG
258: #define RMATCH(ra,rb,rc,rd,re,rf,rg,rw) \
259: { \
260: printf("match() called in line %d \n", __LINE__); \
261: rrc = match(ra,rb,mstart,rc,rd,re,rf,rg,rdepth+1); \
262: printf("to line %d \n", __LINE__); \
263: }
264: #define RRETURN(ra) \
265: { \
266: printf("match() returned %d from line %d ", ra, __LINE__); \
267: return ra; \
268: }
269: #else
270: #define RMATCH(ra,rb,rc,rd,re,rf,rg,rw) \
271: rrc = match(ra,rb,mstart,rc,rd,re,rf,rg,rdepth+1)
272: #define RRETURN(ra) return ra
273: #endif
274:
275: #else
276:
277:
278: /* These versions of the macros manage a private stack on the heap. Note that
279: the "rd" argument of RMATCH isn't actually used in this definition. It's the md
280: argument of match(), which never changes. */
281:

```

```

282: #define REGISTER
283:
284: #define RMATCH(ra,rb,rc,rd,re,rf,rg,rw) \
285: { \
286:  heapframe *newframe = (pcr_stack_malloc)(sizeof(heapframe)); \
287:  frame->Xwhere = rw; \
288:  newframe->Xeptr = ra; \
289:  newframe->Xecode = rb; \
290:  newframe->Xmstart = mstart; \
291:  newframe->Xoffset_top = rc; \
292:  newframe->Xims = re; \
293:  newframe->Xeptrb = rf; \
294:  newframe->Xflags = rg; \
295:  newframe->Xrdepth = frame->Xrdepth + 1; \
296:  newframe->Xprevframe = frame; \
297:  frame = newframe; \
298:  DPRINTF(("restarting from line %d \n", __LINE__)); \
299:  goto HEAP_RECURSE; \
300:  L_##rw: \
301:  DPRINTF(("jumped back to line %d \n", __LINE__)); \
302: }
303:
304: #define RRETURN(ra) \
305: { \
306:  heapframe *newframe = frame; \
307:  frame = newframe->Xprevframe; \
308:  (pcr_stack_free)(newframe); \
309:  if (frame != NULL) \
310:  { \
311:   rrc = ra; \
312:   goto HEAP_RETURN; \
313:  } \
314:  return ra; \
315: }
316:
317:
318: /* Structure for remembering the local variables in a private frame */
319:
320: typedef struct heapframe {
321:  struct heapframe *Xprevframe;
322:

```



```

323: /* Function arguments that may change */
324:
325: const uschar *Xeptr;
326: const uschar *Xecode;
327: const uschar *Xmstart;
328: int Xoffset_top;
329: long int Xims;
330: eptrblock *Xeptrb;
331: int Xflags;
332: unsigned int Xrdepth;
333:
334: /* Function local variables */
335:
336: const uschar *Xcallpat;
337: const uschar *Xcharptr;
338: const uschar *Xdata;
339: const uschar *Xnext;
340: const uschar *Xpp;
341: const uschar *Xprev;
342: const uschar *Xsaved_eptr;
343:
344: recursion_info Xnew_recursive;
345:
346: BOOL Xcur_is_word;
347: BOOL Xcondition;
348: BOOL Xprev_is_word;
349:
350: unsigned long int Xoriginal_ims;
351:
352: #ifdef SUPPORT_UCP
353: int Xprop_type;
354: int Xprop_value;
355: int Xprop_fail_result;
356: int Xprop_category;
357: int Xprop_chartype;
358: int Xprop_script;
359: int Xoclength;
360: uschar Xocchars[8];
361: #endif
362:
363: int Xctype;

```

```

364: unsigned int Xfc;
365: int Xfi;
366: int Xlength;
367: int Xmax;
368: int Xmin;
369: int Xnumber;
370: int Xoffset;
371: int Xop;
372: int Xsave_capture_last;
373: int Xsave_offset1, Xsave_offset2, Xsave_offset3;
374: int Xstacksave[REC_STACK_SAVE_MAX];
375:
376: eptrblock Xnewptrb;
377:
378: /* Where to jump back to */
379:
380: int Xwhere;
381:
382: } heapframe;
383:
384: #endif
385:
386:
387: /*****
388: *****/
389:
390:
391:
392: /*****
393: *      Match from current position      *
394: *****/
395:
396: /* This function is called recursively in many circumstances. Whenever it
397: returns a negative (error) response, the outer incarnation must also return the
398: same response.
399:
400: Performance note: It might be tempting to extract commonly used fields from the
401: md structure (e.g. utf8, end_subject) into individual variables to improve
402: performance. Tests using gcc on a SPARC disproved this; in the first case, it
403: made performance worse.
404:

```

```

405: Arguments:
406:  eptr    pointer to current character in subject
407:  ecode    pointer to current position in compiled code
408:  mstart   pointer to the current match start position (can be modified
409:           by encountering \K)
410:  offset_top  current top pointer
411:  md        pointer to "static" info for the match
412:  ims       current /i, /m, and /s options
413:  eptrb     pointer to chain of blocks containing eptr at start of
414:           brackets - for testing for empty matches
415:  flags     can contain
416:           match_condassert - this is an assertion condition
417:           match_cbegroup - this is the start of an unlimited repeat
418:           group that can match an empty string
419:  rdepth    the recursion depth
420:
421: Returns:   MATCH_MATCH if matched      ) these values are >= 0
422:           MATCH_NOMATCH if failed to match )
423:           a negative PCRE_ERROR_xxx value if aborted by an error condition
424:           (e.g. stopped by repeated call or recursion limit)
425: */
426:
427: static int
428: match(REGISTER USPTR eptr, REGISTER const uschar *ecode, const uschar *mstart,
429: int offset_top, match_data *md, unsigned long int ims, eptrblock *eptrb,
430: int flags, unsigned int rdepth)
431: {
432: /* These variables do not need to be preserved over recursion in this function,
433: so they can be ordinary variables in all cases. Mark some of them with
434: "register" because they are used a lot in loops. */
435:
436: register int rrc;      /* Returns from recursive calls */
437: register int i;        /* Used for loops not involving calls to RMATCH() */
438: register unsigned int c; /* Character values not kept over RMATCH() calls */
439: register BOOL utf8;    /* Local copy of UTF-8 flag for speed */
440:
441: BOOL minimize, possessive; /* Quantifier options */
442:
443: /* When recursion is not being used, all "local" variables that have to be
444: preserved over calls to RMATCH() are part of a "frame" which is obtained from
445: heap storage. Set up the top-level frame here; others are obtained from the

```

```

446: heap whenever RMATCH() does a "recursion". See the macro definitions above. */
447:
448: #ifdef NO_RECURSE
449: heapframe *frame = (pcr_stack_malloc)(sizeof(heapframe));
450: frame->Xprevframe = NULL;      /* Marks the top level */
451:
452: /* Copy in the original argument variables */
453:
454: frame->Xeptr = eptr;
455: frame->Xecode = ecode;
456: frame->Xmstart = mstart;
457: frame->Xoffset_top = offset_top;
458: frame->Xims = ims;
459: frame->Xeptrb = eptrb;
460: frame->Xflags = flags;
461: frame->Xrdepth = rdepth;
462:
463: /* This is where control jumps back to to effect "recursion" */
464:
465: HEAP_RECURSE:
466:
467: /* Macros make the argument variables come from the current frame */
468:
469: #define eptr      frame->Xeptr
470: #define ecode     frame->Xecode
471: #define mstart    frame->Xmstart
472: #define offset_top frame->Xoffset_top
473: #define ims       frame->Xims
474: #define eptrb     frame->Xeptrb
475: #define flags     frame->Xflags
476: #define rdepth    frame->Xrdepth
477:
478: /* Ditto for the local variables */
479:
480: #ifdef SUPPORT_UTF8
481: #define charptr    frame->Xcharptr
482: #endif
483: #define callpat    frame->Xcallpat
484: #define data       frame->Xdata
485: #define next       frame->Xnext
486: #define pp         frame->Xpp

```

```

487: #define prev      frame->Xprev
488: #define saved_eptr  frame->Xsaved_eptr
489:
490: #define new_recursive  frame->Xnew_recursive
491:
492: #define cur_is_word    frame->Xcur_is_word
493: #define condition      frame->Xcondition
494: #define prev_is_word   frame->Xprev_is_word
495:
496: #define original_ims    frame->Xoriginal_ims
497:
498: #ifdef SUPPORT_UCP
499: #define prop_type       frame->Xprop_type
500: #define prop_value      frame->Xprop_value
501: #define prop_fail_result frame->Xprop_fail_result
502: #define prop_category   frame->Xprop_category
503: #define prop_chartype   frame->Xprop_chartype
504: #define prop_script     frame->Xprop_script
505: #define oclength        frame->Xoclength
506: #define occhars         frame->Xocchars
507: #endif
508:
509: #define ctype          frame->Xctype
510: #define fc             frame->Xfc
511: #define fi             frame->Xfi
512: #define length         frame->Xlength
513: #define max            frame->Xmax
514: #define min            frame->Xmin
515: #define number         frame->Xnumber
516: #define offset         frame->Xoffset
517: #define op             frame->Xop
518: #define save_capture_last frame->Xsave_capture_last
519: #define save_offset1    frame->Xsave_offset1
520: #define save_offset2    frame->Xsave_offset2
521: #define save_offset3    frame->Xsave_offset3
522: #define stacksave      frame->Xstacksave
523:
524: #define newptrb        frame->Xnewptrb
525:
526: /* When recursion is being used, local variables are allocated on the stack and
527: get preserved during recursion in the normal way. In this environment, fi and

```

```

528: i, and fc and c, can be the same variables. */
529:
530: #else      /* NO_RECURSE not defined */
531: #define fi i
532: #define fc c
533:
534:
535: #ifdef SUPPORT_UTF8      /* Many of these variables are used only */
536: const uschar *charptr;    /* in small blocks of the code. My normal */
537: #endif      /* style of coding would have declared */
538: const uschar *callpat;    /* them within each of those blocks. */
539: const uschar *data;      /* However, in order to accommodate the */
540: const uschar *next;      /* version of this code that uses an */
541: USPTR pp;      /* external "stack" implemented on the */
542: const uschar *prev;      /* heap, it is easier to declare them all */
543: USPTR saved_eptr;      /* here, so the declarations can be cut */
544:      /* out in a block. The only declarations */
545: recursion_info new_recursive; /* within blocks below are for variables */
546:      /* that do not have to be preserved over */
547: BOOL cur_is_word;      /* a recursive call to RMATCH(). */
548: BOOL condition;
549: BOOL prev_is_word;
550:
551: unsigned long int original_ims;
552:
553: #ifdef SUPPORT_UCP
554: int prop_type;
555: int prop_value;
556: int prop_fail_result;
557: int prop_category;
558: int prop_chartype;
559: int prop_script;
560: int oclength;
561: uschar occhars[8];
562: #endif
563:
564: int ctype;
565: int length;
566: int max;
567: int min;
568: int number;

```

```

569: int offset;
570: int op;
571: int save_capture_last;
572: int save_offset1, save_offset2, save_offset3;
573: int stacksave[REC_STACK_SAVE_MAX];
574:
575: eptrblock newptrb;
576: #endif    /* NO_RECURSE */
577:
578: /* These statements are here to stop the compiler complaining about uninitialized
579: variables. */
580:
581: #ifdef SUPPORT_UCP
582: prop_value = 0;
583: prop_fail_result = 0;
584: #endif
585:
586:
587: /* This label is used for tail recursion, which is used in a few cases even
588: when NO_RECURSE is not defined, in order to reduce the amount of stack that is
589: used. Thanks to Ian Taylor for noticing this possibility and sending the
590: original patch. */
591:
592: TAIL_RECURSE:
593:
594: /* OK, now we can get on with the real code of the function. Recursive calls
595: are specified by the macro RMATCH and RRETURN is used to return. When
596: NO_RECURSE is *not* defined, these just turn into a recursive call to match()
597: and a "return", respectively (possibly with some debugging if DEBUG is
598: defined). However, RMATCH isn't like a function call because it's quite a
599: complicated macro. It has to be used in one particular way. This shouldn't,
600: however, impact performance when true recursion is being used. */
601:
602: #ifdef SUPPORT_UTF8
603: utf8 = md->utf8;    /* Local copy of the flag */
604: #else
605: utf8 = FALSE;
606: #endif
607:
608: /* First check that we haven't called match() too many times, or that we
609: haven't exceeded the recursive call limit. */

```

```

610:
611: if (md->match_call_count++ >= md->match_limit) RRETURN(PCRE_ERROR_MATCHLIMIT);
612: if (rdepth >= md->match_limit_recursion) RRETURN(PCRE_ERROR_RECURSIONLIMIT);
613:
614: original_ims = ims; /* Save for resetting on ')' */
615:
616: /* At the start of a group with an unlimited repeat that may match an empty
617: string, the match_cbegroup flag is set. When this is the case, add the current
618: subject pointer to the chain of such remembered pointers, to be checked when we
619: hit the closing ket, in order to break infinite loops that match no characters.
620: When match() is called in other circumstances, don't add to the chain. The
621: match_cbegroup flag must NOT be used with tail recursion, because the memory
622: block that is used is on the stack, so a new one may be required for each
623: match(). */
624:
625: if ((flags & match_cbegroup) != 0)
626: {
627:   newptrb.epb_saved_eptr = eptr;
628:   newptrb.epb_prev = eptrb;
629:   eptrb = &newptrb;
630: }
631:
632: /* Now start processing the opcodes. */
633:
634: for (;;)
635: {
636:   minimize = possessive = FALSE;
637:   op = *ecode;
638:
639:   /* For partial matching, remember if we ever hit the end of the subject after
640:   matching at least one subject character. */
641:
642:   if (md->partial &&
643:       eptr >= md->end_subject &&
644:       eptr > mstart)
645:     md->hitend = TRUE;
646:
647:   switch(op)
648:   {
649:     case OP_FAIL:
650:       RRETURN(MATCH_NOMATCH);

```



```

651:
652:  case OP_PRUNE:
653:  RMATCH(eptr, ecodes + _pcre_OP_lengths[*ecode], offset_top, md,
654:    ims, eptrb, flags, RM51);
655:  if (rrc != MATCH_NOMATCH) RRETURN(rrc);
656:  RRETURN(MATCH_PRUNE);
657:
658:  case OP_COMMIT:
659:  RMATCH(eptr, ecodes + _pcre_OP_lengths[*ecode], offset_top, md,
660:    ims, eptrb, flags, RM52);
661:  if (rrc != MATCH_NOMATCH) RRETURN(rrc);
662:  RRETURN(MATCH_COMMIT);
663:
664:  case OP_SKIP:
665:  RMATCH(eptr, ecodes + _pcre_OP_lengths[*ecode], offset_top, md,
666:    ims, eptrb, flags, RM53);
667:  if (rrc != MATCH_NOMATCH) RRETURN(rrc);
668:  md->start_match_ptr = eptr; /* Pass back current position */
669:  RRETURN(MATCH_SKIP);
670:
671:  case OP_THEN:
672:  RMATCH(eptr, ecodes + _pcre_OP_lengths[*ecode], offset_top, md,
673:    ims, eptrb, flags, RM54);
674:  if (rrc != MATCH_NOMATCH) RRETURN(rrc);
675:  RRETURN(MATCH_THEN);
676:
677:  /* Handle a capturing bracket. If there is space in the offset vector, save
678:  the current subject position in the working slot at the top of the vector.
679:  We mustn't change the current values of the data slot, because they may be
680:  set from a previous iteration of this group, and be referred to by a
681:  reference inside the group.
682:
683:  If the bracket fails to match, we need to restore this value and also the
684:  values of the final offsets, in case they were set by a previous iteration
685:  of the same bracket.
686:
687:  If there isn't enough space in the offset vector, treat this as if it were
688:  a non-capturing bracket. Don't worry about setting the flag for the error
689:  case here; that is handled in the code for KET. */
690:
691:  case OP_CBRA:

```

```

692: case OP_SCBRA:
693: number = GET2(ecode, 1+LINK_SIZE);
694: offset = number << 1;
695:
696: #ifdef DEBUG
697: printf("start bracket %d \n", number);
698: printf("subject=");
699: pchars(eptr, 16, TRUE, md);
700: printf(" \n");
701: #endif
702:
703: if (offset < md->offset_max)
704: {
705: save_offset1 = md->offset_vector[offset];
706: save_offset2 = md->offset_vector[offset+1];
707: save_offset3 = md->offset_vector[md->offset_end - number];
708: save_capture_last = md->capture_last;
709:
710: DPRINTF(("saving %d %d %d \n", save_offset1, save_offset2, save_offset3));
711: md->offset_vector[md->offset_end - number] = eptr - md->start_subject;
712:
713: flags = (op == OP_SCBRA)? match_cbegroup : 0;
714: do
715: {
716: RMATCH(eptr, ecode + _pcre_OP_lengths[*ecode], offset_top, md,
717: ims, eptrb, flags, RM1);
718: if (rrc != MATCH_NOMATCH && rrc != MATCH_THEN) RRETURN(rrc);
719: md->capture_last = save_capture_last;
720: ecode += GET(ecode, 1);
721: }
722: while (*ecode == OP_ALT);
723:
724: DPRINTF(("bracket %d failed \n", number));
725:
726: md->offset_vector[offset] = save_offset1;
727: md->offset_vector[offset+1] = save_offset2;
728: md->offset_vector[md->offset_end - number] = save_offset3;
729:
730: RRETURN(MATCH_NOMATCH);
731: }
732:

```



```

774:
775:   RMATCH(eptr, ecode + _pcre_OP_lengths[*ecode], offset_top, md, ims,
776:   eptrb, flags, RM2);
777:   if (rrc != MATCH_NOMATCH && rrc != MATCH_THEN) RRETURN(rrc);
778:   ecode += GET(ecode, 1);
779:   }
780:   /* Control never reaches here. */
781:
782:   /* Conditional group: compilation checked that there are no more than
783:   two branches. If the condition is false, skipping the first branch takes us
784:   past the end if there is only one branch, but that's OK because that is
785:   exactly what going to the ket would do. As there is only one branch to be
786:   obeyed, we can use tail recursion to avoid using another stack frame. */
787:
788:   case OP_COND:
789:   case OP_SCOND:
790:   if (ecode[LINK_SIZE+1] == OP_RREF)      /* Recursion test */
791:   {
792:     offset = GET2(ecode, LINK_SIZE + 2); /* Recursion group number*/
793:     condition = md->recursive != NULL &&
794:       (offset == RREF_ANY || offset == md->recursive->group_num);
795:     ecode += condition? 3 : GET(ecode, 1);
796:   }
797:
798:   else if (ecode[LINK_SIZE+1] == OP_CREF) /* Group used test */
799:   {
800:     offset = GET2(ecode, LINK_SIZE+2) << 1; /* Doubled ref number */
801:     condition = offset < offset_top && md->offset_vector[offset] >= 0;
802:     ecode += condition? 3 : GET(ecode, 1);
803:   }
804:
805:   else if (ecode[LINK_SIZE+1] == OP_DEF) /* DEFINE - always false */
806:   {
807:     condition = FALSE;
808:     ecode += GET(ecode, 1);
809:   }
810:
811:   /* The condition is an assertion. Call match() to evaluate it - setting
812:   the final argument match_condassert causes it to stop at the end of an
813:   assertion. */
814:

```

```

815: else
816: {
817:   RMATCH(eptr, ecode + 1 + LINK_SIZE, offset_top, md, ims, NULL,
818:     match_condassert, RM3);
819:   if (rrc == MATCH_MATCH)
820:   {
821:     condition = TRUE;
822:     ecode += 1 + LINK_SIZE + GET(ecode, LINK_SIZE + 2);
823:     while (*ecode == OP_ALT) ecode += GET(ecode, 1);
824:   }
825:   else if (rrc != MATCH_NOMATCH && rrc != MATCH_THEN)
826:   {
827:     RRETURN(rrc);      /* Need braces because of following else */
828:   }
829:   else
830:   {
831:     condition = FALSE;
832:     ecode += GET(ecode, 1);
833:   }
834: }
835:
836: /* We are now at the branch that is to be obeyed. As there is only one,
837: we can use tail recursion to avoid using another stack frame, except when
838: match_cbegroup is required for an unlimited repeat of a possibly empty
839: group. If the second alternative doesn't exist, we can just plough on. */
840:
841: if (condition || *ecode == OP_ALT)
842: {
843:   ecode += 1 + LINK_SIZE;
844:   if (op == OP_SCOND)      /* Possibly empty group */
845:   {
846:     RMATCH(eptr, ecode, offset_top, md, ims, eptrb, match_cbegroup, RM49);
847:     RRETURN(rrc);
848:   }
849:   else                    /* Group must match something */
850:   {
851:     flags = 0;
852:     goto TAIL_RECURSE;
853:   }
854: }
855: else                      /* Condition false & no 2nd alternative */

```

```

856:  {
857:    ecode += 1 + LINK_SIZE;
858:  }
859:  break;
860:
861:
862:  /* End of the pattern, either real or forced. If we are in a top-level
863:  recursion, we should restore the offsets appropriately and continue from
864:  after the call. */
865:
866:  case OP_ACCEPT:
867:  case OP_END:
868:  if (md->recursive != NULL && md->recursive->group_num == 0)
869:  {
870:    recursion_info *rec = md->recursive;
871:    DPRINTF(("End of pattern in a (?0) recursion \n"));
872:    md->recursive = rec->prevrec;
873:    memmove(md->offset_vector, rec->offset_save,
874:      rec->saved_max * sizeof(int));
875:    mstart = rec->save_start;
876:    ims = original_ims;
877:    ecode = rec->after_call;
878:    break;
879:  }
880:
881:  /* Otherwise, if PCRE_NOTEMPTY is set, fail if we have matched an empty
882:  string - backtracking will then try other alternatives, if any. */
883:
884:  if (md->notempty && eptr == mstart) RRETURN(MATCH_NOMATCH);
885:  md->end_match_ptr = eptr;      /* Record where we ended */
886:  md->end_offset_top = offset_top; /* and how many extracts were taken */
887:  md->start_match_ptr = mstart;  /* and the start ( \K can modify) */
888:  RRETURN(MATCH_MATCH);
889:
890:  /* Change option settings */
891:
892:  case OP_OPT:
893:  ims = ecode[1];
894:  ecode += 2;
895:  DPRINTF(("ims set to %02lx \n", ims));
896:  break;

```

```

897:
898:  /* Assertion brackets. Check the alternative branches in turn - the
899:  matching won't pass the KET for an assertion. If any one branch matches,
900:  the assertion is true. Lookbehind assertions have an OP_REVERSE item at the
901:  start of each branch to move the current point backwards, so the code at
902:  this level is identical to the lookahead case. */
903:
904:  case OP_ASSERT:
905:  case OP_ASSERTBACK:
906:  do
907:  {
908:    RMATCH(eptr, ecode + 1 + LINK_SIZE, offset_top, md, ims, NULL, 0,
909:    RM4);
910:    if (rrc == MATCH_MATCH) break;
911:    if (rrc != MATCH_NOMATCH && rrc != MATCH_THEN) RRETURN(rrc);
912:    ecode += GET(ecode, 1);
913:  }
914:  while (*ecode == OP_ALT);
915:  if (*ecode == OP_KET) RRETURN(MATCH_NOMATCH);
916:
917:  /* If checking an assertion for a condition, return MATCH_MATCH. */
918:
919:  if ((flags & match_condassert) != 0) RRETURN(MATCH_MATCH);
920:
921:  /* Continue from after the assertion, updating the offsets high water
922:  mark, since extracts may have been taken during the assertion. */
923:
924:  do ecode += GET(ecode,1); while (*ecode == OP_ALT);
925:  ecode += 1 + LINK_SIZE;
926:  offset_top = md->end_offset_top;
927:  continue;
928:
929:  /* Negative assertion: all branches must fail to match */
930:
931:  case OP_ASSERT_NOT:
932:  case OP_ASSERTBACK_NOT:
933:  do
934:  {
935:    RMATCH(eptr, ecode + 1 + LINK_SIZE, offset_top, md, ims, NULL, 0,
936:    RM5);
937:    if (rrc == MATCH_MATCH) RRETURN(MATCH_NOMATCH);

```

```

938:   if (rrc != MATCH_NOMATCH && rrc != MATCH_THEN) RRETURN(rrc);
939:   ecode += GET(ecode,1);
940:   }
941:   while (*ecode == OP_ALT);
942:
943:   if ((flags & match_condassert) != 0) RRETURN(MATCH_MATCH);
944:
945:   ecode += 1 + LINK_SIZE;
946:   continue;
947:
948:   /* Move the subject pointer back. This occurs only at the start of
949:   each branch of a lookbehind assertion. If we are too close to the start to
950:   move back, this match function fails. When working with UTF-8 we move
951:   back a number of characters, not bytes. */
952:
953:   case OP_REVERSE:
954: #ifdef SUPPORT_UTF8
955:   if (utf8)
956:   {
957:     i = GET(ecode, 1);
958:     while (i-- > 0)
959:     {
960:       eptr--;
961:       if (eptr < md->start_subject) RRETURN(MATCH_NOMATCH);
962:       BACKCHAR(eptr);
963:     }
964:   }
965:   else
966: #endif
967:
968:   /* No UTF-8 support, or not in UTF-8 mode: count is byte count */
969:
970:   {
971:     eptr -= GET(ecode, 1);
972:     if (eptr < md->start_subject) RRETURN(MATCH_NOMATCH);
973:   }
974:
975:   /* Skip to next op code */
976:
977:   ecode += 1 + LINK_SIZE;
978:   break;

```



```

979:
980:  /* The callout item calls an external function, if one is provided, passing
981:  details of the match so far. This is mainly for debugging, though the
982:  function is able to force a failure. */
983:
984:  case OP_CALLOUT:
985:  if (pcre_callout != NULL)
986:  {
987:    pcre_callout_block cb;
988:    cb.version      = 1; /* Version 1 of the callout block */
989:    cb.callout_number = ecode[1];
990:    cb.offset_vector = md->offset_vector;
991:    cb.subject       = (PCRE_SPTR)md->start_subject;
992:    cb.subject_length = md->end_subject - md->start_subject;
993:    cb.start_match    = mstart - md->start_subject;
994:    cb.current_position = eptr - md->start_subject;
995:    cb.pattern_position = GET(ecode, 2);
996:    cb.next_item_length = GET(ecode, 2 + LINK_SIZE);
997:    cb.capture_top     = offset_top/2;
998:    cb.capture_last    = md->capture_last;
999:    cb.callout_data    = md->callout_data;
1000:    if ((rrc = (*pcre_callout)(&cb)) > 0) RRETURN(MATCH_NOMATCH);
1001:    if (rrc < 0) RRETURN(rrc);
1002:  }
1003:  ecode += 2 + 2*LINK_SIZE;
1004:  break;
1005:
1006:  /* Recursion either matches the current regex, or some subexpression. The
1007:  offset data is the offset to the starting bracket from the start of the
1008:  whole pattern. (This is so that it works from duplicated subpatterns.)
1009:
1010:  If there are any capturing brackets started but not finished, we have to
1011:  save their starting points and reinstate them after the recursion. However,
1012:  we don't know how many such there are (offset_top records the completed
1013:  total) so we just have to save all the potential data. There may be up to
1014:  65535 such values, which is too large to put on the stack, but using malloc
1015:  for small numbers seems expensive. As a compromise, the stack is used when
1016:  there are no more than REC_STACK_SAVE_MAX values to store; otherwise malloc
1017:  is used. A problem is what to do if the malloc fails ... there is no way of
1018:  returning to the top level with an error. Save the top REC_STACK_SAVE_MAX
1019:  values on the stack, and accept that the rest may be wrong.

```

```

1020:
1021:  There are also other values that have to be saved. We use a chained
1022:  sequence of blocks that actually live on the stack. Thanks to Robin Houston
1023:  for the original version of this logic. */
1024:
1025:  case OP_RECURSE:
1026:  {
1027:      callpat = md->start_code + GET(ecode, 1);
1028:      new_recursive.group_num = (callpat == md->start_code)? 0 :
1029:          GET2(callpat, 1 + LINK_SIZE);
1030:
1031:      /* Add to "recurring stack" */
1032:
1033:      new_recursive.prevrec = md->recursive;
1034:      md->recursive = &new_recursive;
1035:
1036:      /* Find where to continue from afterwards */
1037:
1038:      ecode += 1 + LINK_SIZE;
1039:      new_recursive.after_call = ecode;
1040:
1041:      /* Now save the offset data. */
1042:
1043:      new_recursive.saved_max = md->offset_end;
1044:      if (new_recursive.saved_max <= REC_STACK_SAVE_MAX)
1045:          new_recursive.offset_save = stacksave;
1046:      else
1047:      {
1048:          new_recursive.offset_save =
1049:              (int *)(pcre_malloc)(new_recursive.saved_max * sizeof(int));
1050:          if (new_recursive.offset_save == NULL) RRETURN(PCRE_ERROR_NOMEMORY);
1051:      }
1052:
1053:      memcpy(new_recursive.offset_save, md->offset_vector,
1054:          new_recursive.saved_max * sizeof(int));
1055:      new_recursive.save_start = mstart;
1056:      mstart = eptr;
1057:
1058:      /* OK, now we can do the recursion. For each top-level alternative we
1059:      restore the offset and recursion data. */
1060:

```

```

1061:     DPRINTF(("Recurring into group %d \n", new_recursive.group_num));
1062:     flags = (*callpat >= OP_SBRA)? match_cbegroup : 0;
1063:     do
1064:     {
1065:         RMATCH(eptr, callpat + _pcre_OP_lengths[*callpat], offset_top,
1066:             md, ims, eptrb, flags, RM6);
1067:         if (rrc == MATCH_MATCH)
1068:         {
1069:             DPRINTF(("Recursion matched \n"));
1070:             md->recursive = new_recursive.prevrec;
1071:             if (new_recursive.offset_save != stacksave)
1072:                 (pcre_free)(new_recursive.offset_save);
1073:             RRETURN(MATCH_MATCH);
1074:         }
1075:         else if (rrc != MATCH_NOMATCH && rrc != MATCH_THEN)
1076:         {
1077:             DPRINTF(("Recursion gave error %d \n", rrc));
1078:             RRETURN(rrc);
1079:         }
1080:
1081:         md->recursive = &new_recursive;
1082:         memcpy(md->offset_vector, new_recursive.offset_save,
1083:             new_recursive.saved_max * sizeof(int));
1084:         callpat += GET(callpat, 1);
1085:     }
1086:     while (*callpat == OP_ALT);
1087:
1088:     DPRINTF(("Recursion didn't match \n"));
1089:     md->recursive = new_recursive.prevrec;
1090:     if (new_recursive.offset_save != stacksave)
1091:         (pcre_free)(new_recursive.offset_save);
1092:     RRETURN(MATCH_NOMATCH);
1093: }
1094: /* Control never reaches here */
1095:
1096: /* "Once" brackets are like assertion brackets except that after a match,
1097: the point in the subject string is not moved back. Thus there can never be
1098: a move back into the brackets. Friedl calls these "atomic" subpatterns.
1099: Check the alternative branches in turn - the matching won't pass the KET
1100: for this kind of subpattern. If any one branch matches, we carry on as at
1101: the end of a normal bracket, leaving the subject pointer. */

```

```

1102:
1103:  case OP_ONCE:
1104:    prev = ecode;
1105:    saved_eptr = eptr;
1106:
1107:    do
1108:      {
1109:        RMATCH(eptr, ecode + 1 + LINK_SIZE, offset_top, md, ims, eptrb, 0, RM7);
1110:        if (rrc == MATCH_MATCH) break;
1111:        if (rrc != MATCH_NOMATCH && rrc != MATCH_THEN) RRETURN(rrc);
1112:        ecode += GET(ecode,1);
1113:      }
1114:    while (*ecode == OP_ALT);
1115:
1116:    /* If hit the end of the group (which could be repeated), fail */
1117:
1118:    if (*ecode != OP_ONCE && *ecode != OP_ALT) RRETURN(MATCH_NOMATCH);
1119:
1120:    /* Continue as from after the assertion, updating the offsets high water
1121:    mark, since extracts may have been taken. */
1122:
1123:    do ecode += GET(ecode, 1); while (*ecode == OP_ALT);
1124:
1125:    offset_top = md->end_offset_top;
1126:    eptr = md->end_match_ptr;
1127:
1128:    /* For a non-repeating ket, just continue at this level. This also
1129:    happens for a repeating ket if no characters were matched in the group.
1130:    This is the forcible breaking of infinite loops as implemented in Perl
1131:    5.005. If there is an options reset, it will get obeyed in the normal
1132:    course of events. */
1133:
1134:    if (*ecode == OP_KET || eptr == saved_eptr)
1135:      {
1136:        ecode += 1+LINK_SIZE;
1137:        break;
1138:      }
1139:
1140:    /* The repeating kets try the rest of the pattern or restart from the
1141:    preceding bracket, in the appropriate order. The second "call" of match()
1142:    uses tail recursion, to avoid using another stack frame. We need to reset

```

```

1143: any options that changed within the bracket before re-running it, so
1144: check the next opcode. */
1145:
1146: if (ecode[1+LINK_SIZE] == OP_OPT)
1147: {
1148:     ims = (ims & ~PCRE_IMS) | ecode[4];
1149:     DPRINTF(("ims set to %02lx at group repeat \n", ims));
1150: }
1151:
1152: if (*ecode == OP_KETRMIN)
1153: {
1154:     RMATCH(eptr, ecode + 1 + LINK_SIZE, offset_top, md, ims, eptrb, 0, RM8);
1155:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1156:     ecode = prev;
1157:     flags = 0;
1158:     goto TAIL_RECURSE;
1159: }
1160: else /* OP_KETRMAX */
1161: {
1162:     RMATCH(eptr, prev, offset_top, md, ims, eptrb, match_cbegroup, RM9);
1163:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1164:     ecode += 1 + LINK_SIZE;
1165:     flags = 0;
1166:     goto TAIL_RECURSE;
1167: }
1168: /* Control never gets here */
1169:
1170: /* An alternation is the end of a branch; scan along to find the end of the
1171: bracketed group and go to there. */
1172:
1173: case OP_ALT:
1174: do ecode += GET(ecode,1); while (*ecode == OP_ALT);
1175: break;
1176:
1177: /* BRAZERO, BRAMINZERO and SKIPZERO occur just before a bracket group,
1178: indicating that it may occur zero times. It may repeat infinitely, or not
1179: at all - i.e. it could be ()* or ()? or even (){0} in the pattern. Brackets
1180: with fixed upper repeat limits are compiled as a number of copies, with the
1181: optional ones preceded by BRAZERO or BRAMINZERO. */
1182:
1183: case OP_BRAZERO:

```

```

1184:  {
1185:    next = ecode+1;
1186:    RMATCH(eptr, next, offset_top, md, ims, eptrb, 0, RM10);
1187:    if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1188:    do next += GET(next,1); while (*next == OP_ALT);
1189:    ecode = next + 1 + LINK_SIZE;
1190:  }
1191:  break;
1192:
1193:  case OP_BRAMINZERO:
1194:    {
1195:      next = ecode+1;
1196:      do next += GET(next, 1); while (*next == OP_ALT);
1197:      RMATCH(eptr, next + 1+LINK_SIZE, offset_top, md, ims, eptrb, 0, RM11);
1198:      if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1199:      ecode++;
1200:    }
1201:    break;
1202:
1203:  case OP_SKIPZERO:
1204:    {
1205:      next = ecode+1;
1206:      do next += GET(next,1); while (*next == OP_ALT);
1207:      ecode = next + 1 + LINK_SIZE;
1208:    }
1209:    break;
1210:
1211:    /* End of a group, repeated or non-repeating. */
1212:
1213:  case OP_KET:
1214:  case OP_KETRMIN:
1215:  case OP_KETRMAX:
1216:    prev = ecode - GET(ecode, 1);
1217:
1218:    /* If this was a group that remembered the subject start, in order to break
1219:    infinite repeats of empty string matches, retrieve the subject start from
1220:    the chain. Otherwise, set it NULL. */
1221:
1222:    if (*prev >= OP_SBRA)
1223:    {
1224:      saved_eptr = eptrb->epb_saved_eptr; /* Value at start of group */

```

```

1225:  eptrb = eptrb->epb_prev;          /* Backup to previous group */
1226:  }
1227:  else saved_eptr = NULL;
1228:
1229:  /* If we are at the end of an assertion group, stop matching and return
1230:  MATCH_MATCH, but record the current high water mark for use by positive
1231:  assertions. Do this also for the "once" (atomic) groups. */
1232:
1233:  if (*prev == OP_ASSERT || *prev == OP_ASSERT_NOT ||
1234:      *prev == OP_ASSERTBACK || *prev == OP_ASSERTBACK_NOT ||
1235:      *prev == OP_ONCE)
1236:  {
1237:      md->end_match_ptr = eptr;      /* For ONCE */
1238:      md->end_offset_top = offset_top;
1239:      RRETURN(MATCH_MATCH);
1240:  }
1241:
1242:  /* For capturing groups we have to check the group number back at the start
1243:  and if necessary complete handling an extraction by setting the offsets and
1244:  bumping the high water mark. Note that whole-pattern recursion is coded as
1245:  a recurse into group 0, so it won't be picked up here. Instead, we catch it
1246:  when the OP_END is reached. Other recursion is handled here. */
1247:
1248:  if (*prev == OP_CBRA || *prev == OP_SCBRA)
1249:  {
1250:      number = GET2(prev, 1+LINK_SIZE);
1251:      offset = number << 1;
1252:
1253:  #ifdef DEBUG
1254:      printf("end bracket %d", number);
1255:      printf("\n");
1256:  #endif
1257:
1258:      md->capture_last = number;
1259:      if (offset >= md->offset_max) md->offset_overflow = TRUE; else
1260:      {
1261:          md->offset_vector[offset] =
1262:              md->offset_vector[md->offset_end - number];
1263:          md->offset_vector[offset+1] = eptr - md->start_subject;
1264:          if (offset_top <= offset) offset_top = offset + 2;
1265:      }

```

```

1266:
1267:  /* Handle a recursively called group. Restore the offsets
1268:  appropriately and continue from after the call. */
1269:
1270:  if (md->recursive != NULL && md->recursive->group_num == number)
1271:  {
1272:      recursion_info *rec = md->recursive;
1273:      DPRINTF(("Recursion (%d) succeeded - continuing \n", number));
1274:      md->recursive = rec->prevrec;
1275:      mstart = rec->save_start;
1276:      memcpy(md->offset_vector, rec->offset_save,
1277:          rec->saved_max * sizeof(int));
1278:      ecode = rec->after_call;
1279:      ims = original_ims;
1280:      break;
1281:  }
1282:  }
1283:
1284:  /* For both capturing and non-capturing groups, reset the value of the ims
1285:  flags, in case they got changed during the group. */
1286:
1287:  ims = original_ims;
1288:  DPRINTF(("ims reset to %02lx \n", ims));
1289:
1290:  /* For a non-repeating ket, just continue at this level. This also
1291:  happens for a repeating ket if no characters were matched in the group.
1292:  This is the forcible breaking of infinite loops as implemented in Perl
1293:  5.005. If there is an options reset, it will get obeyed in the normal
1294:  course of events. */
1295:
1296:  if (*ecode == OP_KET || eptr == saved_eptr)
1297:  {
1298:      ecode += 1 + LINK_SIZE;
1299:      break;
1300:  }
1301:
1302:  /* The repeating kets try the rest of the pattern or restart from the
1303:  preceding bracket, in the appropriate order. In the second case, we can use
1304:  tail recursion to avoid using another stack frame, unless we have an
1305:  unlimited repeat of a group that can match an empty string. */
1306:

```



```

1307: flags = (*prev >= OP_SBRA)? match_cbegroup : 0;
1308:
1309: if (*ecode == OP_KETRMIN)
1310: {
1311:   RMATCH(eptr, ecode + 1 + LINK_SIZE, offset_top, md, ims, eptrb, 0, RM12);
1312:   if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1313:   if (flags != 0) /* Could match an empty string */
1314:   {
1315:     RMATCH(eptr, prev, offset_top, md, ims, eptrb, flags, RM50);
1316:     RRETURN(rrc);
1317:   }
1318:   ecode = prev;
1319:   goto TAIL_RECURSE;
1320: }
1321: else /* OP_KETRMAX */
1322: {
1323:   RMATCH(eptr, prev, offset_top, md, ims, eptrb, flags, RM13);
1324:   if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1325:   ecode += 1 + LINK_SIZE;
1326:   flags = 0;
1327:   goto TAIL_RECURSE;
1328: }
1329: /* Control never gets here */
1330:
1331: /* Start of subject unless notbol, or after internal newline if multiline */
1332:
1333: case OP_CIRC:
1334:   if (md->notbol && eptr == md->start_subject) RRETURN(MATCH_NOMATCH);
1335:   if ((ims & PCRE_MULTILINE) != 0)
1336:   {
1337:     if (eptr != md->start_subject &&
1338:         (eptr == md->end_subject || !WAS_NEWLINE(eptr)))
1339:       RRETURN(MATCH_NOMATCH);
1340:     ecode++;
1341:     break;
1342:   }
1343:   /* ... else fall through */
1344:
1345:   /* Start of subject assertion */
1346:
1347:   case OP_SOD:

```

```

1348:  if (eptr != md->start_subject) RRETURN(MATCH_NOMATCH);
1349:  ecode++;
1350:  break;
1351:
1352:  /* Start of match assertion */
1353:
1354:  case OP_SOM:
1355:  if (eptr != md->start_subject + md->start_offset) RRETURN(MATCH_NOMATCH);
1356:  ecode++;
1357:  break;
1358:
1359:  /* Reset the start of match point */
1360:
1361:  case OP_SET_SOM:
1362:  mstart = eptr;
1363:  ecode++;
1364:  break;
1365:
1366:  /* Assert before internal newline if multiline, or before a terminating
1367:  newline unless endonly is set, else end of subject unless noteol is set. */
1368:
1369:  case OP_DOLL:
1370:  if ((ims & PCRE_MULTILINE) != 0)
1371:  {
1372:    if (eptr < md->end_subject)
1373:      { if (!IS_NEWLINE(eptr)) RRETURN(MATCH_NOMATCH); }
1374:    else
1375:      { if (md->noteol) RRETURN(MATCH_NOMATCH); }
1376:    ecode++;
1377:    break;
1378:  }
1379:  else
1380:  {
1381:    if (md->noteol) RRETURN(MATCH_NOMATCH);
1382:    if (!md->endonly)
1383:      {
1384:        if (eptr != md->end_subject &&
1385:            (!IS_NEWLINE(eptr) || eptr != md->end_subject - md->nllen))
1386:          RRETURN(MATCH_NOMATCH);
1387:        ecode++;
1388:        break;

```

```

1389:     }
1390: }
1391: /* ... else fall through for endonly */
1392:
1393: /* End of subject assertion ( \z) */
1394:
1395: case OP_EOD:
1396: if (eptr < md->end_subject) RRETURN(MATCH_NOMATCH);
1397: ecode++;
1398: break;
1399:
1400: /* End of subject or ending \n assertion ( \Z) */
1401:
1402: case OP_EODN:
1403: if (eptr != md->end_subject &&
1404:     (!IS_NEWLINE(eptr) || eptr != md->end_subject - md->nllen))
1405:     RRETURN(MATCH_NOMATCH);
1406: ecode++;
1407: break;
1408:
1409: /* Word boundary assertions */
1410:
1411: case OP_NOT_WORD_BOUNDARY:
1412: case OP_WORD_BOUNDARY:
1413: {
1414:
1415: /* Find out if the previous and current characters are "word" characters.
1416: It takes a bit more work in UTF-8 mode. Characters > 255 are assumed to
1417: be "non-word" characters. */
1418:
1419: #ifdef SUPPORT_UTF8
1420: if (utf8)
1421: {
1422: if (eptr == md->start_subject) prev_is_word = FALSE; else
1423: {
1424: const uschar *lastptr = eptr - 1;
1425: while((*lastptr & 0xc0) == 0x80) lastptr--;
1426: GETCHAR(c, lastptr);
1427: prev_is_word = c < 256 && (md->ctypes[c] & ctype_word) != 0;
1428: }
1429: if (eptr >= md->end_subject) cur_is_word = FALSE; else

```

```

1430:     {
1431:         GETCHAR(c, eptr);
1432:         cur_is_word = c < 256 && (md->ctypes[c] & ctype_word) != 0;
1433:     }
1434: }
1435: else
1436: #endif
1437:
1438: /* More streamlined when not in UTF-8 mode */
1439:
1440: {
1441:     prev_is_word = (eptr != md->start_subject) &&
1442:         ((md->ctypes[eptr[-1]] & ctype_word) != 0);
1443:     cur_is_word = (eptr < md->end_subject) &&
1444:         ((md->ctypes[*eptr] & ctype_word) != 0);
1445: }
1446:
1447: /* Now see if the situation is what we want */
1448:
1449: if ((*ecode++ == OP_WORD_BOUNDARY)?
1450:     cur_is_word == prev_is_word : cur_is_word != prev_is_word)
1451:     RRETURN(MATCH_NOMATCH);
1452: }
1453: break;
1454:
1455: /* Match a single character type; inline for speed */
1456:
1457: case OP_ANY:
1458:     if (IS_NEWLINE(eptr)) RRETURN(MATCH_NOMATCH);
1459:     /* Fall through */
1460:
1461: case OP_ALLANY:
1462:     if (eptr++ >= md->end_subject) RRETURN(MATCH_NOMATCH);
1463:     if (utf8) while (eptr < md->end_subject && (*eptr & 0xc0) == 0x80) eptr++;
1464:     ecode++;
1465:     break;
1466:
1467: /* Match a single byte, even in UTF-8 mode. This opcode really does match
1468: any byte, even newline, independent of the setting of PCRE_DOTALL. */
1469:
1470: case OP_ANYBYTE:

```

```

1471:  if (eptr++ >= md->end_subject) RRETURN(MATCH_NOMATCH);
1472:  ecode++;
1473:  break;
1474:
1475:  case OP_NOT_DIGIT:
1476:  if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1477:  GETCHARINCTEST(c, eptr);
1478:  if (
1479: #ifdef SUPPORT_UTF8
1480:     c < 256 &&
1481: #endif
1482:     (md->ctypes[c] & ctype_digit) != 0
1483:  )
1484:  RRETURN(MATCH_NOMATCH);
1485:  ecode++;
1486:  break;
1487:
1488:  case OP_DIGIT:
1489:  if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1490:  GETCHARINCTEST(c, eptr);
1491:  if (
1492: #ifdef SUPPORT_UTF8
1493:     c >= 256 ||
1494: #endif
1495:     (md->ctypes[c] & ctype_digit) == 0
1496:  )
1497:  RRETURN(MATCH_NOMATCH);
1498:  ecode++;
1499:  break;
1500:
1501:  case OP_NOT_WHITESPACE:
1502:  if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1503:  GETCHARINCTEST(c, eptr);
1504:  if (
1505: #ifdef SUPPORT_UTF8
1506:     c < 256 &&
1507: #endif
1508:     (md->ctypes[c] & ctype_space) != 0
1509:  )
1510:  RRETURN(MATCH_NOMATCH);
1511:  ecode++;

```

```

1512: break;
1513:
1514: case OP_WHITESPACE:
1515:   if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1516:   GETCHARINCTEST(c, eptr);
1517:   if (
1518: #ifdef SUPPORT_UTF8
1519:     c >= 256 ||
1520: #endif
1521:     (md->ctypes[c] & ctype_space) == 0
1522:   )
1523:     RRETURN(MATCH_NOMATCH);
1524:   ecode++;
1525:   break;
1526:
1527: case OP_NOT_WORDCHAR:
1528:   if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1529:   GETCHARINCTEST(c, eptr);
1530:   if (
1531: #ifdef SUPPORT_UTF8
1532:     c < 256 &&
1533: #endif
1534:     (md->ctypes[c] & ctype_word) != 0
1535:   )
1536:     RRETURN(MATCH_NOMATCH);
1537:   ecode++;
1538:   break;
1539:
1540: case OP_WORDCHAR:
1541:   if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1542:   GETCHARINCTEST(c, eptr);
1543:   if (
1544: #ifdef SUPPORT_UTF8
1545:     c >= 256 ||
1546: #endif
1547:     (md->ctypes[c] & ctype_word) == 0
1548:   )
1549:     RRETURN(MATCH_NOMATCH);
1550:   ecode++;
1551:   break;
1552:

```

```

1553: case OP_ANYNL:
1554: if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1555: GETCHARINCTEST(c, eptr);
1556: switch(c)
1557: {
1558: default: RRETURN(MATCH_NOMATCH);
1559: case 0x000d:
1560: if (eptr < md->end_subject && *eptr == 0x0a) eptr++;
1561: break;
1562:
1563: case 0x000a:
1564: break;
1565:
1566: case 0x000b:
1567: case 0x000c:
1568: case 0x0085:
1569: case 0x2028:
1570: case 0x2029:
1571: if (md->bsr_anycrlf) RRETURN(MATCH_NOMATCH);
1572: break;
1573: }
1574: ecode++;
1575: break;
1576:
1577: case OP_NOT_HSPACE:
1578: if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1579: GETCHARINCTEST(c, eptr);
1580: switch(c)
1581: {
1582: default: break;
1583: case 0x09: /* HT */
1584: case 0x20: /* SPACE */
1585: case 0xa0: /* NBSP */
1586: case 0x1680: /* OGHAM SPACE MARK */
1587: case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
1588: case 0x2000: /* EN QUAD */
1589: case 0x2001: /* EM QUAD */
1590: case 0x2002: /* EN SPACE */
1591: case 0x2003: /* EM SPACE */
1592: case 0x2004: /* THREE-PER-EM SPACE */
1593: case 0x2005: /* FOUR-PER-EM SPACE */

```

```

1594: case 0x2006: /* SIX-PER-EM SPACE */
1595: case 0x2007: /* FIGURE SPACE */
1596: case 0x2008: /* PUNCTUATION SPACE */
1597: case 0x2009: /* THIN SPACE */
1598: case 0x200A: /* HAIR SPACE */
1599: case 0x202f: /* NARROW NO-BREAK SPACE */
1600: case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
1601: case 0x3000: /* IDEOGRAPHIC SPACE */
1602: RRETURN(MATCH_NOMATCH);
1603: }
1604: ecode++;
1605: break;
1606:
1607: case OP_HSPACE:
1608: if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1609: GETCHARINCTEST(c, eptr);
1610: switch(c)
1611: {
1612: default: RRETURN(MATCH_NOMATCH);
1613: case 0x09: /* HT */
1614: case 0x20: /* SPACE */
1615: case 0xa0: /* NBSP */
1616: case 0x1680: /* OGHAM SPACE MARK */
1617: case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
1618: case 0x2000: /* EN QUAD */
1619: case 0x2001: /* EM QUAD */
1620: case 0x2002: /* EN SPACE */
1621: case 0x2003: /* EM SPACE */
1622: case 0x2004: /* THREE-PER-EM SPACE */
1623: case 0x2005: /* FOUR-PER-EM SPACE */
1624: case 0x2006: /* SIX-PER-EM SPACE */
1625: case 0x2007: /* FIGURE SPACE */
1626: case 0x2008: /* PUNCTUATION SPACE */
1627: case 0x2009: /* THIN SPACE */
1628: case 0x200A: /* HAIR SPACE */
1629: case 0x202f: /* NARROW NO-BREAK SPACE */
1630: case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
1631: case 0x3000: /* IDEOGRAPHIC SPACE */
1632: break;
1633: }
1634: ecode++;

```



```

1635: break;
1636:
1637: case OP_NOT_VSPACE:
1638: if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1639: GETCHARINCTEST(c, eptr);
1640: switch(c)
1641: {
1642: default: break;
1643: case 0x0a: /* LF */
1644: case 0x0b: /* VT */
1645: case 0x0c: /* FF */
1646: case 0x0d: /* CR */
1647: case 0x85: /* NEL */
1648: case 0x2028: /* LINE SEPARATOR */
1649: case 0x2029: /* PARAGRAPH SEPARATOR */
1650: RRETURN(MATCH_NOMATCH);
1651: }
1652: ecode++;
1653: break;
1654:
1655: case OP_VSPACE:
1656: if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1657: GETCHARINCTEST(c, eptr);
1658: switch(c)
1659: {
1660: default: RRETURN(MATCH_NOMATCH);
1661: case 0x0a: /* LF */
1662: case 0x0b: /* VT */
1663: case 0x0c: /* FF */
1664: case 0x0d: /* CR */
1665: case 0x85: /* NEL */
1666: case 0x2028: /* LINE SEPARATOR */
1667: case 0x2029: /* PARAGRAPH SEPARATOR */
1668: break;
1669: }
1670: ecode++;
1671: break;
1672:
1673: #ifdef SUPPORT_UCP
1674: /* Check the next character by Unicode property. We will get here only
1675: if the support is in the binary; otherwise a compile-time error occurs. */

```

```

1676:
1677: case OP_PROP:
1678: case OP_NOTPROP:
1679: if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1680: GETCHARINCTEST(c, eptr);
1681: {
1682:     const ucd_record * prop = GET_UCD(c);
1683:
1684:     switch(ecode[1])
1685:     {
1686:     case PT_ANY:
1687:         if (op == OP_NOTPROP) RRETURN(MATCH_NOMATCH);
1688:         break;
1689:
1690:     case PT_LAMP:
1691:         if ((prop->chartype == ucp_Lu ||
1692:             prop->chartype == ucp_Ll ||
1693:             prop->chartype == ucp_Lt) == (op == OP_NOTPROP))
1694:             RRETURN(MATCH_NOMATCH);
1695:         break;
1696:
1697:     case PT_GC:
1698:         if ((ecode[2] != _pcre_ucp_gentype[prop->chartype]) == (op == OP_PROP))
1699:             RRETURN(MATCH_NOMATCH);
1700:         break;
1701:
1702:     case PT_PC:
1703:         if ((ecode[2] != prop->chartype) == (op == OP_PROP))
1704:             RRETURN(MATCH_NOMATCH);
1705:         break;
1706:
1707:     case PT_SC:
1708:         if ((ecode[2] != prop->script) == (op == OP_PROP))
1709:             RRETURN(MATCH_NOMATCH);
1710:         break;
1711:
1712:     default:
1713:         RRETURN(PCRE_ERROR_INTERNAL);
1714:     }
1715:
1716:     ecode += 3;

```

```

1717:     }
1718:     break;
1719:
1720:     /* Match an extended Unicode sequence. We will get here only if the support
1721:     is in the binary; otherwise a compile-time error occurs. */
1722:
1723:     case OP_EXTUNI:
1724:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1725:     GETCHARINCTEST(c, eptr);
1726:     {
1727:     int category = UCD_CATEGORY(c);
1728:     if (category == ucp_M) RRETURN(MATCH_NOMATCH);
1729:     while (eptr < md->end_subject)
1730:     {
1731:     int len = 1;
1732:     if (!utf8) c = *eptr; else
1733:     {
1734:     GETCHARLEN(c, eptr, len);
1735:     }
1736:     category = UCD_CATEGORY(c);
1737:     if (category != ucp_M) break;
1738:     eptr += len;
1739:     }
1740:     }
1741:     ecode++;
1742:     break;
1743: #endif
1744:
1745:
1746:     /* Match a back reference, possibly repeatedly. Look past the end of the
1747:     item to see if there is repeat information following. The code is similar
1748:     to that for character classes, but repeated for efficiency. Then obey
1749:     similar code to character type repeats - written out again for speed.
1750:     However, if the referenced string is the empty string, always treat
1751:     it as matched, any number of times (otherwise there could be infinite
1752:     loops). */
1753:
1754:     case OP_REF:
1755:     {
1756:     offset = GET2(ecode, 1) << 1;          /* Doubled ref number */
1757:     ecode += 3;

```

```

1758:
1759:  /* If the reference is unset, there are two possibilities:
1760:
1761:  (a) In the default, Perl-compatible state, set the length to be longer
1762:  than the amount of subject left; this ensures that every attempt at a
1763:  match fails. We can't just fail here, because of the possibility of
1764:  quantifiers with zero minima.
1765:
1766:  (b) If the JavaScript compatibility flag is set, set the length to zero
1767:  so that the back reference matches an empty string.
1768:
1769:  Otherwise, set the length to the length of what was matched by the
1770:  referenced subpattern. */
1771:
1772:  if (offset >= offset_top || md->offset_vector[offset] < 0)
1773:      length = (md->jscript_compat)? 0 : md->end_subject - eptr + 1;
1774:  else
1775:      length = md->offset_vector[offset+1] - md->offset_vector[offset];
1776:
1777:  /* Set up for repetition, or handle the non-repeated case */
1778:
1779:  switch (*ecode)
1780:  {
1781:      case OP_CRSTAR:
1782:      case OP_CRMINSTAR:
1783:      case OP_CRPLUS:
1784:      case OP_CRMINPLUS:
1785:      case OP_CRQUERY:
1786:      case OP_CRMINQUERY:
1787:          c = *ecode++ - OP_CRSTAR;
1788:          minimize = (c & 1) != 0;
1789:          min = rep_min[c];          /* Pick up values from tables; */
1790:          max = rep_max[c];          /* zero for max => infinity */
1791:          if (max == 0) max = INT_MAX;
1792:          break;
1793:
1794:      case OP_CRRANGE:
1795:      case OP_CRMINRANGE:
1796:          minimize = (*ecode == OP_CRMINRANGE);
1797:          min = GET2(ecode, 1);
1798:          max = GET2(ecode, 3);

```

```

1799:     if (max == 0) max = INT_MAX;
1800:     ecode += 5;
1801:     break;
1802:
1803:     default:      /* No repeat follows */
1804:     if (!match_ref(offset, eptr, length, md, ims)) RRETURN(MATCH_NOMATCH);
1805:     eptr += length;
1806:     continue;      /* With the main loop */
1807: }
1808:
1809: /* If the length of the reference is zero, just continue with the
1810: main loop. */
1811:
1812: if (length == 0) continue;
1813:
1814: /* First, ensure the minimum number of matches are present. We get back
1815: the length of the reference string explicitly rather than passing the
1816: address of eptr, so that eptr can be a register variable. */
1817:
1818: for (i = 1; i <= min; i++)
1819: {
1820:     if (!match_ref(offset, eptr, length, md, ims)) RRETURN(MATCH_NOMATCH);
1821:     eptr += length;
1822: }
1823:
1824: /* If min = max, continue at the same level without recursion.
1825: They are not both allowed to be zero. */
1826:
1827: if (min == max) continue;
1828:
1829: /* If minimizing, keep trying and advancing the pointer */
1830:
1831: if (minimize)
1832: {
1833:     for (fi = min;; fi++)
1834:     {
1835:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM14);
1836:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1837:         if (fi >= max || !match_ref(offset, eptr, length, md, ims))
1838:             RRETURN(MATCH_NOMATCH);
1839:         eptr += length;

```

```

1840:     }
1841:     /* Control never gets here */
1842: }
1843:
1844: /* If maximizing, find the longest string and work backwards */
1845:
1846: else
1847: {
1848:     pp = eptr;
1849:     for (i = min; i < max; i++)
1850:     {
1851:         if (!match_ref(offset, eptr, length, md, ims)) break;
1852:         eptr += length;
1853:     }
1854:     while (eptr >= pp)
1855:     {
1856:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM15);
1857:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1858:         eptr -= length;
1859:     }
1860:     RRETURN(MATCH_NOMATCH);
1861: }
1862: }
1863: /* Control never gets here */
1864:
1865:
1866:
1867: /* Match a bit-mapped character class, possibly repeatedly. This op code is
1868: used when all the characters in the class have values in the range 0-255,
1869: and either the matching is careful, or the characters are in the range
1870: 0-127 when UTF-8 processing is enabled. The only difference between
1871: OP_CLASS and OP_NCLASS occurs when a data character outside the range is
1872: encountered.
1873:
1874: First, look past the end of the item to see if there is repeat information
1875: following. Then obey similar code to character type repeats - written out
1876: again for speed. */
1877:
1878: case OP_NCLASS:
1879: case OP_CLASS:
1880: {

```

```

1881:  data = ecode + 1;          /* Save for matching */
1882:  ecode += 33;              /* Advance past the item */
1883:
1884:  switch (*ecode)
1885:  {
1886:      case OP_CRSTAR:
1887:      case OP_CRMINSTAR:
1888:      case OP_CRPLUS:
1889:      case OP_CRMINPLUS:
1890:      case OP_CRQUERY:
1891:      case OP_CRMINQUERY:
1892:      c = *ecode++ - OP_CRSTAR;
1893:      minimize = (c & 1) != 0;
1894:      min = rep_min[c];      /* Pick up values from tables; */
1895:      max = rep_max[c];      /* zero for max => infinity */
1896:      if (max == 0) max = INT_MAX;
1897:      break;
1898:
1899:      case OP_CRRANGE:
1900:      case OP_CRMINRANGE:
1901:      minimize = (*ecode == OP_CRMINRANGE);
1902:      min = GET2(ecode, 1);
1903:      max = GET2(ecode, 3);
1904:      if (max == 0) max = INT_MAX;
1905:      ecode += 5;
1906:      break;
1907:
1908:      default:              /* No repeat follows */
1909:      min = max = 1;
1910:      break;
1911:  }
1912:
1913:  /* First, ensure the minimum number of matches are present. */
1914:
1915:  #ifdef SUPPORT_UTF8
1916:      /* UTF-8 mode */
1917:      if (utf8)
1918:      {
1919:          for (i = 1; i <= min; i++)
1920:          {
1921:              if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);

```

```

1922:    GETCHARINC(c, eptr);
1923:    if (c > 255)
1924:        {
1925:            if (op == OP_CLASS) RRETURN(MATCH_NOMATCH);
1926:        }
1927:    else
1928:        {
1929:            if (((data[c/8] & (1 << (c&7))) == 0) RRETURN(MATCH_NOMATCH);
1930:        }
1931:    }
1932:    }
1933:    else
1934: #endif
1935:    /* Not UTF-8 mode */
1936:    {
1937:        for (i = 1; i <= min; i++)
1938:            {
1939:                if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1940:                c = *eptr++;
1941:                if (((data[c/8] & (1 << (c&7))) == 0) RRETURN(MATCH_NOMATCH);
1942:            }
1943:        }
1944:
1945:    /* If max == min we can continue with the main loop without the
1946:       need to recurse. */
1947:
1948:    if (min == max) continue;
1949:
1950:    /* If minimizing, keep testing the rest of the expression and advancing
1951:       the pointer while it matches the class. */
1952:
1953:    if (minimize)
1954:        {
1955: #ifdef SUPPORT_UTF8
1956:         /* UTF-8 mode */
1957:         if (utf8)
1958:             {
1959:                 for (fi = min;; fi++)
1960:                     {
1961:                         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM16);
1962:                         if (rrc != MATCH_NOMATCH) RRETURN(rrc);

```



```

1963:     if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1964:     GETCHARINC(c, eptr);
1965:     if (c > 255)
1966:     {
1967:         if (op == OP_CLASS) RRETURN(MATCH_NOMATCH);
1968:     }
1969:     else
1970:     {
1971:         if ((data[c/8] & (1 << (c&7))) == 0) RRETURN(MATCH_NOMATCH);
1972:     }
1973:     }
1974:     }
1975:     else
1976: #endif
1977:     /* Not UTF-8 mode */
1978:     {
1979:         for (fi = min;; fi++)
1980:         {
1981:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM17);
1982:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
1983:             if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
1984:             c = *eptr++;
1985:             if ((data[c/8] & (1 << (c&7))) == 0) RRETURN(MATCH_NOMATCH);
1986:         }
1987:     }
1988:     /* Control never gets here */
1989:     }
1990:
1991:     /* If maximizing, find the longest possible run, then work backwards. */
1992:
1993:     else
1994:     {
1995:         pp = eptr;
1996:
1997: #ifdef SUPPORT_UTF8
1998:         /* UTF-8 mode */
1999:         if (utf8)
2000:         {
2001:             for (i = min; i < max; i++)
2002:             {
2003:                 int len = 1;

```

```

2004:     if (eptr >= md->end_subject) break;
2005:     GETCHARLEN(c, eptr, len);
2006:     if (c > 255)
2007:     {
2008:         if (op == OP_CLASS) break;
2009:     }
2010:     else
2011:     {
2012:         if (((data[c/8] & (1 << (c&7))) == 0) break;
2013:     }
2014:     eptr += len;
2015: }
2016: for (;;)
2017: {
2018:     RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM18);
2019:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2020:     if (eptr-- == pp) break;    /* Stop if tried at original pos */
2021:     BACKCHAR(eptr);
2022: }
2023: }
2024: else
2025: #endif
2026: /* Not UTF-8 mode */
2027: {
2028:     for (i = min; i < max; i++)
2029:     {
2030:         if (eptr >= md->end_subject) break;
2031:         c = *eptr;
2032:         if (((data[c/8] & (1 << (c&7))) == 0) break;
2033:         eptr++;
2034:     }
2035:     while (eptr >= pp)
2036:     {
2037:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM19);
2038:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2039:         eptr--;
2040:     }
2041: }
2042:
2043: RRETURN(MATCH_NOMATCH);
2044: }

```

```

2045:     }
2046:     /* Control never gets here */
2047:
2048:
2049:     /* Match an extended character class. This opcode is encountered only
2050:     in UTF-8 mode, because that's the only time it is compiled. */
2051:
2052: #ifdef SUPPORT_UTF8
2053:     case OP_XCLASS:
2054:     {
2055:         data = ecode + 1 + LINK_SIZE;          /* Save for matching */
2056:         ecode += GET(ecode, 1);                 /* Advance past the item */
2057:
2058:         switch (*ecode)
2059:         {
2060:             case OP_CRSTAR:
2061:             case OP_CRMINSTAR:
2062:             case OP_CRPLUS:
2063:             case OP_CRMINPLUS:
2064:             case OP_CRQUERY:
2065:             case OP_CRMINQUERY:
2066:                 c = *ecode++ - OP_CRSTAR;
2067:                 minimize = (c & 1) != 0;
2068:                 min = rep_min[c];               /* Pick up values from tables; */
2069:                 max = rep_max[c];               /* zero for max => infinity */
2070:                 if (max == 0) max = INT_MAX;
2071:                 break;
2072:
2073:             case OP_CRRANGE:
2074:             case OP_CRMINRANGE:
2075:                 minimize = (*ecode == OP_CRMINRANGE);
2076:                 min = GET2(ecode, 1);
2077:                 max = GET2(ecode, 3);
2078:                 if (max == 0) max = INT_MAX;
2079:                 ecode += 5;
2080:                 break;
2081:
2082:             default:                /* No repeat follows */
2083:                 min = max = 1;
2084:                 break;
2085:         }

```

```

2086:
2087:  /* First, ensure the minimum number of matches are present. */
2088:
2089:  for (i = 1; i <= min; i++)
2090:  {
2091:    if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2092:    GETCHARINC(c, eptr);
2093:    if (!_pcre_xclass(c, data)) RRETURN(MATCH_NOMATCH);
2094:  }
2095:
2096:  /* If max == min we can continue with the main loop without the
2097:  need to recurse. */
2098:
2099:  if (min == max) continue;
2100:
2101:  /* If minimizing, keep testing the rest of the expression and advancing
2102:  the pointer while it matches the class. */
2103:
2104:  if (minimize)
2105:  {
2106:    for (fi = min;; fi++)
2107:    {
2108:      RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM20);
2109:      if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2110:      if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2111:      GETCHARINC(c, eptr);
2112:      if (!_pcre_xclass(c, data)) RRETURN(MATCH_NOMATCH);
2113:    }
2114:    /* Control never gets here */
2115:  }
2116:
2117:  /* If maximizing, find the longest possible run, then work backwards. */
2118:
2119:  else
2120:  {
2121:    pp = eptr;
2122:    for (i = min; i < max; i++)
2123:    {
2124:      int len = 1;
2125:      if (eptr >= md->end_subject) break;
2126:      GETCHARLEN(c, eptr, len);

```

```

2127:     if (!_pcre_xclass(c, data)) break;
2128:     eptr += len;
2129: }
2130: for(;;)
2131: {
2132:     RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM21);
2133:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2134:     if (eptr-- == pp) break;    /* Stop if tried at original pos */
2135:     if (utf8) BACKCHAR(eptr);
2136: }
2137: RRETURN(MATCH_NOMATCH);
2138: }
2139:
2140: /* Control never gets here */
2141: }
2142: #endif /* End of XCLASS */
2143:
2144: /* Match a single character, casefully */
2145:
2146: case OP_CHAR:
2147: #ifdef SUPPORT_UTF8
2148:     if (utf8)
2149:     {
2150:         length = 1;
2151:         ecode++;
2152:         GETCHARLEN(fc, ecode, length);
2153:         if (length > md->end_subject - eptr) RRETURN(MATCH_NOMATCH);
2154:         while (length-- > 0) if (*ecode++ != *eptr++) RRETURN(MATCH_NOMATCH);
2155:     }
2156:     else
2157: #endif
2158:
2159: /* Non-UTF-8 mode */
2160:     {
2161:         if (md->end_subject - eptr < 1) RRETURN(MATCH_NOMATCH);
2162:         if (ecode[1] != *eptr++) RRETURN(MATCH_NOMATCH);
2163:         ecode += 2;
2164:     }
2165:     break;
2166:
2167: /* Match a single character, caselessly */

```

```

2168:
2169:  case OP_CHARNC:
2170: #ifdef SUPPORT_UTF8
2171:  if (utf8)
2172:  {
2173:    length = 1;
2174:    ecode++;
2175:    GETCHARLEN(fc, ecode, length);
2176:
2177:    if (length > md->end_subject - eptr) RRETURN(MATCH_NOMATCH);
2178:
2179:    /* If the pattern character's value is < 128, we have only one byte, and
2180:    can use the fast lookup table. */
2181:
2182:    if (fc < 128)
2183:    {
2184:      if (md->lcc[*ecode++] != md->lcc[*eptr++]) RRETURN(MATCH_NOMATCH);
2185:    }
2186:
2187:    /* Otherwise we must pick up the subject character */
2188:
2189:    else
2190:    {
2191:      unsigned int dc;
2192:      GETCHARINC(dc, eptr);
2193:      ecode += length;
2194:
2195:      /* If we have Unicode property support, we can use it to test the other
2196:      case of the character, if there is one. */
2197:
2198:      if (fc != dc)
2199:      {
2200: #ifdef SUPPORT_UCP
2201:        if (dc != UCD_OTHERCASE(fc))
2202: #endif
2203:          RRETURN(MATCH_NOMATCH);
2204:      }
2205:    }
2206:  }
2207:  else
2208: #endif /* SUPPORT_UTF8 */

```

```

2209:
2210: /* Non-UTF-8 mode */
2211: {
2212:   if (md->end_subject - eptr < 1) RRETURN(MATCH_NOMATCH);
2213:   if (md->lcc[ecode[1]] != md->lcc[*eptr++]) RRETURN(MATCH_NOMATCH);
2214:   ecode += 2;
2215: }
2216: break;
2217:
2218: /* Match a single character repeatedly. */
2219:
2220: case OP_EXACT:
2221:   min = max = GET2(ecode, 1);
2222:   ecode += 3;
2223:   goto REPEATCHAR;
2224:
2225: case OP_POSUPTO:
2226:   possessive = TRUE;
2227:   /* Fall through */
2228:
2229: case OP_UPTO:
2230: case OP_MINUPTO:
2231:   min = 0;
2232:   max = GET2(ecode, 1);
2233:   minimize = *ecode == OP_MINUPTO;
2234:   ecode += 3;
2235:   goto REPEATCHAR;
2236:
2237: case OP_POSSTAR:
2238:   possessive = TRUE;
2239:   min = 0;
2240:   max = INT_MAX;
2241:   ecode++;
2242:   goto REPEATCHAR;
2243:
2244: case OP_POSPLUS:
2245:   possessive = TRUE;
2246:   min = 1;
2247:   max = INT_MAX;
2248:   ecode++;
2249:   goto REPEATCHAR;

```

```

2250:
2251:  case OP_POSQUERY:
2252:    possessive = TRUE;
2253:    min = 0;
2254:    max = 1;
2255:    ecode++;
2256:    goto REPEATCHAR;
2257:
2258:  case OP_STAR:
2259:  case OP_MINSTAR:
2260:  case OP_PLUS:
2261:  case OP_MINPLUS:
2262:  case OP_QUERY:
2263:  case OP_MINQUERY:
2264:    c = *ecode++ - OP_STAR;
2265:    minimize = (c & 1) != 0;
2266:    min = rep_min[c];          /* Pick up values from tables; */
2267:    max = rep_max[c];          /* zero for max => infinity */
2268:    if (max == 0) max = INT_MAX;
2269:
2270:    /* Common code for all repeated single-character matches. We can give
2271:    up quickly if there are fewer than the minimum number of characters left in
2272:    the subject. */
2273:
2274:  REPEATCHAR:
2275:  #ifdef SUPPORT_UTF8
2276:    if (utf8)
2277:    {
2278:      length = 1;
2279:      charptr = ecode;
2280:      GETCHARLEN(fc, ecode, length);
2281:      if (min * length > md->end_subject - eptr) RRETURN(MATCH_NOMATCH);
2282:      ecode += length;
2283:
2284:      /* Handle multibyte character matching specially here. There is
2285:      support for caseless matching if UCP support is present. */
2286:
2287:      if (length > 1)
2288:      {
2289:  #ifdef SUPPORT_UCP
2290:      unsigned int othercase;

```



```

2291:     if ((ims & PCRE_CASELESS) != 0 &&
2292:         (othercase = UCD_OTHERCASE(fc)) != fc)
2293:         oclength = _pcre_ord2utf8(othercase, occhars);
2294:     else oclength = 0;
2295: #endif /* SUPPORT_UCP */
2296:
2297:     for (i = 1; i <= min; i++)
2298:     {
2299:         if (memcmp(eptr, charptr, length) == 0) eptr += length;
2300: #ifdef SUPPORT_UCP
2301:         /* Need braces because of following else */
2302:         else if (oclength == 0) { RRETURN(MATCH_NOMATCH); }
2303:         else
2304:         {
2305:             if (memcmp(eptr, occhars, oclength) != 0) RRETURN(MATCH_NOMATCH);
2306:             eptr += oclength;
2307:         }
2308: #else /* without SUPPORT_UCP */
2309:         else { RRETURN(MATCH_NOMATCH); }
2310: #endif /* SUPPORT_UCP */
2311:     }
2312:
2313:     if (min == max) continue;
2314:
2315:     if (minimize)
2316:     {
2317:         for (fi = min;; fi++)
2318:         {
2319:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM22);
2320:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2321:             if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2322:             if (memcmp(eptr, charptr, length) == 0) eptr += length;
2323: #ifdef SUPPORT_UCP
2324:             /* Need braces because of following else */
2325:             else if (oclength == 0) { RRETURN(MATCH_NOMATCH); }
2326:             else
2327:             {
2328:                 if (memcmp(eptr, occhars, oclength) != 0) RRETURN(MATCH_NOMATCH);
2329:                 eptr += oclength;
2330:             }
2331: #else /* without SUPPORT_UCP */

```

```

2332:     else { RRETURN (MATCH_NOMATCH); }
2333: #endif /* SUPPORT_UCP */
2334:     }
2335:     /* Control never gets here */
2336:     }
2337:
2338:     else /* Maximize */
2339:     {
2340:         pp = eptr;
2341:         for (i = min; i < max; i++)
2342:         {
2343:             if (eptr > md->end_subject - length) break;
2344:             if (memcmp(eptr, charptr, length) == 0) eptr += length;
2345: #ifdef SUPPORT_UCP
2346:             else if (oclength == 0) break;
2347:             else
2348:             {
2349:                 if (memcmp(eptr, occhars, oclength) != 0) break;
2350:                 eptr += oclength;
2351:             }
2352: #else /* without SUPPORT_UCP */
2353:             else break;
2354: #endif /* SUPPORT_UCP */
2355:         }
2356:
2357:         if (possessive) continue;
2358:         for(;;)
2359:         {
2360:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM23);
2361:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2362:             if (eptr == pp) RRETURN(MATCH_NOMATCH);
2363: #ifdef SUPPORT_UCP
2364:             eptr--;
2365:             BACKCHAR(eptr);
2366: #else /* without SUPPORT_UCP */
2367:             eptr -= length;
2368: #endif /* SUPPORT_UCP */
2369:         }
2370:     }
2371:     /* Control never gets here */
2372:     }

```

```

2373:
2374:  /* If the length of a UTF-8 character is 1, we fall through here, and
2375:  obey the code as for non-UTF-8 characters below, though in this case the
2376:  value of fc will always be < 128. */
2377:  }
2378:  else
2379: #endif /* SUPPORT_UTF8 */
2380:
2381:  /* When not in UTF-8 mode, load a single-byte character. */
2382:  {
2383:    if (min > md->end_subject - eptr) RRETURN(MATCH_NOMATCH);
2384:    fc = *ecode++;
2385:  }
2386:
2387:  /* The value of fc at this point is always less than 256, though we may or
2388:  may not be in UTF-8 mode. The code is duplicated for the caseless and
2389:  careful cases, for speed, since matching characters is likely to be quite
2390:  common. First, ensure the minimum number of matches are present. If min =
2391:  max, continue at the same level without recursing. Otherwise, if
2392:  minimizing, keep trying the rest of the expression and advancing one
2393:  matching character if failing, up to the maximum. Alternatively, if
2394:  maximizing, find the maximum number of characters and work backwards. */
2395:
2396:  DPRINTF(("matching %c{%d,%d} against subject %.*s\n", fc, min, max,
2397:    max, eptr));
2398:
2399:  if ((ims & PCRE_CASELESS) != 0)
2400:  {
2401:    fc = md->lcc[fc];
2402:    for (i = 1; i <= min; i++)
2403:      if (fc != md->lcc[*eptr++]) RRETURN(MATCH_NOMATCH);
2404:    if (min == max) continue;
2405:    if (minimize)
2406:    {
2407:      for (fi = min;; fi++)
2408:      {
2409:        RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM24);
2410:        if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2411:        if (fi >= max || eptr >= md->end_subject ||
2412:            fc != md->lcc[*eptr++])
2413:          RRETURN(MATCH_NOMATCH);

```

```

2414:     }
2415:     /* Control never gets here */
2416:     }
2417: else /* Maximize */
2418:     {
2419:         pp = eptr;
2420:         for (i = min; i < max; i++)
2421:         {
2422:             if (eptr >= md->end_subject || fc != md->lcc[*eptr]) break;
2423:             eptr++;
2424:         }
2425:         if (possessive) continue;
2426:         while (eptr >= pp)
2427:         {
2428:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM25);
2429:             eptr--;
2430:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2431:         }
2432:         RRETURN(MATCH_NOMATCH);
2433:     }
2434:     /* Control never gets here */
2435:     }
2436:
2437: /* Caseful comparisons (includes all multi-byte characters) */
2438:
2439: else
2440:     {
2441:         for (i = 1; i <= min; i++) if (fc != *eptr++) RRETURN(MATCH_NOMATCH);
2442:         if (min == max) continue;
2443:         if (minimize)
2444:         {
2445:             for (fi = min;; fi++)
2446:             {
2447:                 RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM26);
2448:                 if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2449:                 if (fi >= max || eptr >= md->end_subject || fc != *eptr++)
2450:                     RRETURN(MATCH_NOMATCH);
2451:             }
2452:             /* Control never gets here */
2453:         }
2454:         else /* Maximize */

```

```

2455:     {
2456:     pp = eptr;
2457:     for (i = min; i < max; i++)
2458:     {
2459:         if (eptr >= md->end_subject || fc != *eptr) break;
2460:         eptr++;
2461:     }
2462:     if (possessive) continue;
2463:     while (eptr >= pp)
2464:     {
2465:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM27);
2466:         eptr--;
2467:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2468:     }
2469:     RRETURN(MATCH_NOMATCH);
2470:     }
2471:     }
2472:     /* Control never gets here */
2473:
2474:     /* Match a negated single one-byte character. The character we are
2475:     checking can be multibyte. */
2476:
2477:     case OP_NOT:
2478:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2479:     ecode++;
2480:     GETCHARINCTEST(c, eptr);
2481:     if ((ims & PCRE_CASELESS) != 0)
2482:     {
2483:     #ifdef SUPPORT_UTF8
2484:         if (c < 256)
2485:     #endif
2486:         c = md->lcc[c];
2487:         if (md->lcc[*ecode++] == c) RRETURN(MATCH_NOMATCH);
2488:     }
2489:     else
2490:     {
2491:         if (*ecode++ == c) RRETURN(MATCH_NOMATCH);
2492:     }
2493:     break;
2494:
2495:     /* Match a negated single one-byte character repeatedly. This is almost a

```

```

2496: repeat of the code for a repeated single character, but I haven't found a
2497: nice way of commoning these up that doesn't require a test of the
2498: positive/negative option for each character match. Maybe that wouldn't add
2499: very much to the time taken, but character matching *is* what this is all
2500: about... */
2501:
2502: case OP_NOTEXACT:
2503: min = max = GET2(ecode, 1);
2504: ecode += 3;
2505: goto REPEATNOTCHAR;
2506:
2507: case OP_NOTUPTO:
2508: case OP_NOTMINUPTO:
2509: min = 0;
2510: max = GET2(ecode, 1);
2511: minimize = *ecode == OP_NOTMINUPTO;
2512: ecode += 3;
2513: goto REPEATNOTCHAR;
2514:
2515: case OP_NOTPOSSTAR:
2516: possessive = TRUE;
2517: min = 0;
2518: max = INT_MAX;
2519: ecode++;
2520: goto REPEATNOTCHAR;
2521:
2522: case OP_NOTPOSPLUS:
2523: possessive = TRUE;
2524: min = 1;
2525: max = INT_MAX;
2526: ecode++;
2527: goto REPEATNOTCHAR;
2528:
2529: case OP_NOTPOSQUERY:
2530: possessive = TRUE;
2531: min = 0;
2532: max = 1;
2533: ecode++;
2534: goto REPEATNOTCHAR;
2535:
2536: case OP_NOTPOSUPTO:

```

```

2537:    possessive = TRUE;
2538:    min = 0;
2539:    max = GET2(ecode, 1);
2540:    ecode += 3;
2541:    goto REPEATNOTCHAR;
2542:
2543:    case OP_NOTSTAR:
2544:    case OP_NOTMINSTAR:
2545:    case OP_NOTPLUS:
2546:    case OP_NOTMINPLUS:
2547:    case OP_NOTQUERY:
2548:    case OP_NOTMINQUERY:
2549:    c = *ecode++ - OP_NOTSTAR;
2550:    minimize = (c & 1) != 0;
2551:    min = rep_min[c];           /* Pick up values from tables; */
2552:    max = rep_max[c];           /* zero for max => infinity */
2553:    if (max == 0) max = INT_MAX;
2554:
2555:    /* Common code for all repeated single-byte matches. We can give up quickly
2556:    if there are fewer than the minimum number of bytes left in the
2557:    subject. */
2558:
2559:    REPEATNOTCHAR:
2560:    if (min > md->end_subject - eptr) RRETURN(MATCH_NOMATCH);
2561:    fc = *ecode++;
2562:
2563:    /* The code is duplicated for the caseless and careful cases, for speed,
2564:    since matching characters is likely to be quite common. First, ensure the
2565:    minimum number of matches are present. If min = max, continue at the same
2566:    level without recursing. Otherwise, if minimizing, keep trying the rest of
2567:    the expression and advancing one matching character if failing, up to the
2568:    maximum. Alternatively, if maximizing, find the maximum number of
2569:    characters and work backwards. */
2570:
2571:    DPRINTF(("negative matching %c{%d,%d} against subject %.*s\n", fc, min, max,
2572:    max, eptr));
2573:
2574:    if ((ims & PCRE_CASELESS) != 0)
2575:    {
2576:        fc = md->lcc[fc];
2577:

```

```

2578: #ifdef SUPPORT_UTF8
2579:     /* UTF-8 mode */
2580:     if (utf8)
2581:     {
2582:         register unsigned int d;
2583:         for (i = 1; i <= min; i++)
2584:         {
2585:             GETCHARINC(d, eptr);
2586:             if (d < 256) d = md->lcc[d];
2587:             if (fc == d) RRETURN(MATCH_NOMATCH);
2588:         }
2589:     }
2590:     else
2591: #endif
2592:
2593:     /* Not UTF-8 mode */
2594:     {
2595:         for (i = 1; i <= min; i++)
2596:             if (fc == md->lcc[*eptr++]) RRETURN(MATCH_NOMATCH);
2597:     }
2598:
2599:     if (min == max) continue;
2600:
2601:     if (minimize)
2602:     {
2603: #ifdef SUPPORT_UTF8
2604:         /* UTF-8 mode */
2605:         if (utf8)
2606:         {
2607:             register unsigned int d;
2608:             for (fi = min;; fi++)
2609:             {
2610:                 RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM28);
2611:                 if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2612:                 if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2613:                 GETCHARINC(d, eptr);
2614:                 if (d < 256) d = md->lcc[d];
2615:                 if (fc == d) RRETURN(MATCH_NOMATCH);
2616:             }
2617:         }
2618:     }

```



```

2619:     else
2620: #endif
2621:     /* Not UTF-8 mode */
2622:     {
2623:         for (fi = min;; fi++)
2624:         {
2625:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM29);
2626:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2627:             if (fi >= max || eptr >= md->end_subject || fc == md->lcc[*eptr++])
2628:                 RRETURN(MATCH_NOMATCH);
2629:         }
2630:     }
2631:     /* Control never gets here */
2632: }
2633:
2634: /* Maximize case */
2635:
2636: else
2637: {
2638:     pp = eptr;
2639:
2640: #ifdef SUPPORT_UTF8
2641:     /* UTF-8 mode */
2642:     if (utf8)
2643:     {
2644:         register unsigned int d;
2645:         for (i = min; i < max; i++)
2646:         {
2647:             int len = 1;
2648:             if (eptr >= md->end_subject) break;
2649:             GETCHARLEN(d, eptr, len);
2650:             if (d < 256) d = md->lcc[d];
2651:             if (fc == d) break;
2652:             eptr += len;
2653:         }
2654:         if (possessive) continue;
2655:         for(;;)
2656:         {
2657:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM30);
2658:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2659:             if (eptr-- == pp) break;      /* Stop if tried at original pos */

```

```

2660:     BACKCHAR(eptr);
2661: }
2662: }
2663: else
2664: #endif
2665:     /* Not UTF-8 mode */
2666:     {
2667:         for (i = min; i < max; i++)
2668:         {
2669:             if (eptr >= md->end_subject || fc == md->lcc[*eptr]) break;
2670:             eptr++;
2671:         }
2672:         if (possessive) continue;
2673:         while (eptr >= pp)
2674:         {
2675:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM31);
2676:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2677:             eptr--;
2678:         }
2679:     }
2680:
2681:     RRETURN(MATCH_NOMATCH);
2682: }
2683: /* Control never gets here */
2684: }
2685:
2686: /* Caseful comparisons */
2687:
2688: else
2689:     {
2690: #ifdef SUPPORT_UTF8
2691:     /* UTF-8 mode */
2692:     if (utf8)
2693:     {
2694:         register unsigned int d;
2695:         for (i = 1; i <= min; i++)
2696:         {
2697:             GETCHARINC(d, eptr);
2698:             if (fc == d) RRETURN(MATCH_NOMATCH);
2699:         }
2700:     }

```

```

2701:     else
2702: #endif
2703:     /* Not UTF-8 mode */
2704:     {
2705:         for (i = 1; i <= min; i++)
2706:             if (fc == *eptr++) RRETURN(MATCH_NOMATCH);
2707:     }
2708:
2709:     if (min == max) continue;
2710:
2711:     if (minimize)
2712:     {
2713: #ifdef SUPPORT_UTF8
2714:         /* UTF-8 mode */
2715:         if (utf8)
2716:         {
2717:             register unsigned int d;
2718:             for (fi = min;; fi++)
2719:             {
2720:                 RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM32);
2721:                 if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2722:                 if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2723:                 GETCHARINC(d, eptr);
2724:                 if (fc == d) RRETURN(MATCH_NOMATCH);
2725:             }
2726:         }
2727:     }
2728: #endif
2729:     /* Not UTF-8 mode */
2730:     {
2731:         for (fi = min;; fi++)
2732:         {
2733:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM33);
2734:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2735:             if (fi >= max || eptr >= md->end_subject || fc == *eptr++)
2736:                 RRETURN(MATCH_NOMATCH);
2737:         }
2738:     }
2739:     /* Control never gets here */
2740: }
2741:

```

```

2742:  /* Maximize case */
2743:
2744:  else
2745:  {
2746:      pp = eptr;
2747:
2748:  #ifdef SUPPORT_UTF8
2749:      /* UTF-8 mode */
2750:      if (utf8)
2751:      {
2752:          register unsigned int d;
2753:          for (i = min; i < max; i++)
2754:          {
2755:              int len = 1;
2756:              if (eptr >= md->end_subject) break;
2757:              GETCHARLEN(d, eptr, len);
2758:              if (fc == d) break;
2759:              eptr += len;
2760:          }
2761:          if (possessive) continue;
2762:          for(;;)
2763:          {
2764:              RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM34);
2765:              if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2766:              if (eptr-- == pp) break;      /* Stop if tried at original pos */
2767:              BACKCHAR(eptr);
2768:          }
2769:      }
2770:  else
2771:  #endif
2772:      /* Not UTF-8 mode */
2773:      {
2774:          for (i = min; i < max; i++)
2775:          {
2776:              if (eptr >= md->end_subject || fc == *eptr) break;
2777:              eptr++;
2778:          }
2779:          if (possessive) continue;
2780:          while (eptr >= pp)
2781:          {
2782:              RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM35);

```

```

2783:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);
2784:     eptr--;
2785: }
2786: }
2787:
2788: RRETURN(MATCH_NOMATCH);
2789: }
2790: }
2791: /* Control never gets here */
2792:
2793: /* Match a single character type repeatedly; several different opcodes
2794: share code. This is very similar to the code for single characters, but we
2795: repeat it in the interests of efficiency. */
2796:
2797: case OP_TYPEEXACT:
2798: min = max = GET2(ecode, 1);
2799: minimize = TRUE;
2800: ecode += 3;
2801: goto REPEATTYPE;
2802:
2803: case OP_TYPEUPTO:
2804: case OP_TYPEMINUPTO:
2805: min = 0;
2806: max = GET2(ecode, 1);
2807: minimize = *ecode == OP_TYPEMINUPTO;
2808: ecode += 3;
2809: goto REPEATTYPE;
2810:
2811: case OP_TYPEPOSSTAR:
2812: possessive = TRUE;
2813: min = 0;
2814: max = INT_MAX;
2815: ecode++;
2816: goto REPEATTYPE;
2817:
2818: case OP_TYPEPOSPLUS:
2819: possessive = TRUE;
2820: min = 1;
2821: max = INT_MAX;
2822: ecode++;
2823: goto REPEATTYPE;

```

```

2824:
2825:  case OP_TYPEPOSQUERY:
2826:    possessive = TRUE;
2827:    min = 0;
2828:    max = 1;
2829:    ecode++;
2830:    goto REPEATTYPE;
2831:
2832:  case OP_TYPEPOSUPTO:
2833:    possessive = TRUE;
2834:    min = 0;
2835:    max = GET2(ecode, 1);
2836:    ecode += 3;
2837:    goto REPEATTYPE;
2838:
2839:  case OP_TYPESTAR:
2840:  case OP_TYPEMINSTAR:
2841:  case OP_TYPEPLUS:
2842:  case OP_TYPEMINPLUS:
2843:  case OP_TYPEQUERY:
2844:  case OP_TYPEMINQUERY:
2845:    c = *ecode++ - OP_TYPESTAR;
2846:    minimize = (c & 1) != 0;
2847:    min = rep_min[c];          /* Pick up values from tables; */
2848:    max = rep_max[c];          /* zero for max => infinity */
2849:    if (max == 0) max = INT_MAX;
2850:
2851:    /* Common code for all repeated single character type matches. Note that
2852:    in UTF-8 mode, '.' matches a character of any length, but for the other
2853:    character types, the valid characters are all one-byte long. */
2854:
2855:  REPEATTYPE:
2856:    ctype = *ecode++;          /* Code for the character type */
2857:
2858:  #ifdef SUPPORT_UCP
2859:    if (ctype == OP_PROP || ctype == OP_NOTPROP)
2860:    {
2861:      prop_fail_result = ctype == OP_NOTPROP;
2862:      prop_type = *ecode++;
2863:      prop_value = *ecode++;
2864:    }

```

```

2865:     else prop_type = -1;
2866: #endif
2867:
2868:     /* First, ensure the minimum number of matches are present. Use inline
2869:     code for maximizing the speed, and do the type test once at the start
2870:     (i.e. keep it out of the loop). Also we can test that there are at least
2871:     the minimum number of bytes before we start. This isn't as effective in
2872:     UTF-8 mode, but it does no harm. Separate the UTF-8 code completely as that
2873:     is tidier. Also separate the UCP code, which can be the same for both UTF-8
2874:     and single-bytes. */
2875:
2876:     if (min > md->end_subject - eptr) RRETURN(MATCH_NOMATCH);
2877:     if (min > 0)
2878:     {
2879: #ifdef SUPPORT_UCP
2880:         if (prop_type >= 0)
2881:         {
2882:             switch(prop_type)
2883:             {
2884:                 case PT_ANY:
2885:                     if (prop_fail_result) RRETURN(MATCH_NOMATCH);
2886:                     for (i = 1; i <= min; i++)
2887:                     {
2888:                         if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2889:                         GETCHARINCTEST(c, eptr);
2890:                     }
2891:                     break;
2892:
2893:                 case PT_LAMP:
2894:                     for (i = 1; i <= min; i++)
2895:                     {
2896:                         if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2897:                         GETCHARINCTEST(c, eptr);
2898:                         prop_chartype = UCD_CHARTYPE(c);
2899:                         if ((prop_chartype == ucp_Lu ||
2900:                             prop_chartype == ucp_Ll ||
2901:                             prop_chartype == ucp_Lt) == prop_fail_result)
2902:                             RRETURN(MATCH_NOMATCH);
2903:                     }
2904:                     break;
2905:

```

```

2906:     case PT_GC:
2907:         for (i = 1; i <= min; i++)
2908:             {
2909:                 if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2910:                 GETCHARINCTEST(c, eptr);
2911:                 prop_category = UCD_CATEGORY(c);
2912:                 if ((prop_category == prop_value) == prop_fail_result)
2913:                     RRETURN(MATCH_NOMATCH);
2914:             }
2915:         break;
2916:
2917:     case PT_PC:
2918:         for (i = 1; i <= min; i++)
2919:             {
2920:                 if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2921:                 GETCHARINCTEST(c, eptr);
2922:                 prop_chartype = UCD_CHARTYPE(c);
2923:                 if ((prop_chartype == prop_value) == prop_fail_result)
2924:                     RRETURN(MATCH_NOMATCH);
2925:             }
2926:         break;
2927:
2928:     case PT_SC:
2929:         for (i = 1; i <= min; i++)
2930:             {
2931:                 if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2932:                 GETCHARINCTEST(c, eptr);
2933:                 prop_script = UCD_SCRIPT(c);
2934:                 if ((prop_script == prop_value) == prop_fail_result)
2935:                     RRETURN(MATCH_NOMATCH);
2936:             }
2937:         break;
2938:
2939:     default:
2940:         RRETURN(PCRE_ERROR_INTERNAL);
2941:     }
2942: }
2943:
2944: /* Match extended Unicode sequences. We will get here only if the
2945: support is in the binary; otherwise a compile-time error occurs. */
2946:

```



```

2947:     else if (ctype == OP_EXTUNI)
2948:     {
2949:         for (i = 1; i <= min; i++)
2950:         {
2951:             GETCHARINCTEST(c, eptr);
2952:             prop_category = UCD_CATEGORY(c);
2953:             if (prop_category == ucp_M) RRETURN(MATCH_NOMATCH);
2954:             while (eptr < md->end_subject)
2955:             {
2956:                 int len = 1;
2957:                 if (!utf8) c = *eptr; else
2958:                 {
2959:                     GETCHARLEN(c, eptr, len);
2960:                 }
2961:                 prop_category = UCD_CATEGORY(c);
2962:                 if (prop_category != ucp_M) break;
2963:                 eptr += len;
2964:             }
2965:         }
2966:     }
2967:
2968:     else
2969: #endif /* SUPPORT_UCP */
2970:
2971: /* Handle all other cases when the coding is UTF-8 */
2972:
2973: #ifdef SUPPORT_UTF8
2974:     if (utf8) switch(ctype)
2975:     {
2976:         case OP_ANY:
2977:             for (i = 1; i <= min; i++)
2978:             {
2979:                 if (eptr >= md->end_subject || IS_NEWLINE(eptr))
2980:                     RRETURN(MATCH_NOMATCH);
2981:                 eptr++;
2982:                 while (eptr < md->end_subject && (*eptr & 0xc0) == 0x80) eptr++;
2983:             }
2984:             break;
2985:
2986:         case OP_ALLANY:
2987:             for (i = 1; i <= min; i++)

```

```

2988:     {
2989:         if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
2990:         eptr++;
2991:         while (eptr < md->end_subject && (*eptr & 0xc0) == 0x80) eptr++;
2992:     }
2993:     break;
2994:
2995:     case OP_ANYBYTE:
2996:         eptr += min;
2997:         break;
2998:
2999:     case OP_ANYNL:
3000:         for (i = 1; i <= min; i++)
3001:         {
3002:             if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3003:             GETCHARINC(c, eptr);
3004:             switch(c)
3005:             {
3006:                 default: RRETURN(MATCH_NOMATCH);
3007:                 case 0x000d:
3008:                     if (eptr < md->end_subject && *eptr == 0x0a) eptr++;
3009:                     break;
3010:
3011:                 case 0x000a:
3012:                     break;
3013:
3014:                 case 0x000b:
3015:                 case 0x000c:
3016:                 case 0x0085:
3017:                 case 0x2028:
3018:                 case 0x2029:
3019:                     if (md->bsr_anycrlf) RRETURN(MATCH_NOMATCH);
3020:                     break;
3021:             }
3022:         }
3023:         break;
3024:
3025:     case OP_NOT_HSPACE:
3026:         for (i = 1; i <= min; i++)
3027:         {
3028:             if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);

```

```

3029:   GETCHARINC(c, eptr);
3030:   switch(c)
3031:   {
3032:       default: break;
3033:       case 0x09:   /* HT */
3034:       case 0x20:   /* SPACE */
3035:       case 0xa0:   /* NBSP */
3036:       case 0x1680: /* OGHAM SPACE MARK */
3037:       case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
3038:       case 0x2000: /* EN QUAD */
3039:       case 0x2001: /* EM QUAD */
3040:       case 0x2002: /* EN SPACE */
3041:       case 0x2003: /* EM SPACE */
3042:       case 0x2004: /* THREE-PER-EM SPACE */
3043:       case 0x2005: /* FOUR-PER-EM SPACE */
3044:       case 0x2006: /* SIX-PER-EM SPACE */
3045:       case 0x2007: /* FIGURE SPACE */
3046:       case 0x2008: /* PUNCTUATION SPACE */
3047:       case 0x2009: /* THIN SPACE */
3048:       case 0x200A: /* HAIR SPACE */
3049:       case 0x202f: /* NARROW NO-BREAK SPACE */
3050:       case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
3051:       case 0x3000: /* IDEOGRAPHIC SPACE */
3052:       RRETURN(MATCH_NOMATCH);
3053:   }
3054: }
3055: break;
3056:
3057: case OP_HSPACE:
3058: for (i = 1; i <= min; i++)
3059: {
3060: if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3061: GETCHARINC(c, eptr);
3062: switch(c)
3063: {
3064:     default: RRETURN(MATCH_NOMATCH);
3065:     case 0x09:   /* HT */
3066:     case 0x20:   /* SPACE */
3067:     case 0xa0:   /* NBSP */
3068:     case 0x1680: /* OGHAM SPACE MARK */
3069:     case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */

```

```

3070:     case 0x2000: /* EN QUAD */
3071:     case 0x2001: /* EM QUAD */
3072:     case 0x2002: /* EN SPACE */
3073:     case 0x2003: /* EM SPACE */
3074:     case 0x2004: /* THREE-PER-EM SPACE */
3075:     case 0x2005: /* FOUR-PER-EM SPACE */
3076:     case 0x2006: /* SIX-PER-EM SPACE */
3077:     case 0x2007: /* FIGURE SPACE */
3078:     case 0x2008: /* PUNCTUATION SPACE */
3079:     case 0x2009: /* THIN SPACE */
3080:     case 0x200A: /* HAIR SPACE */
3081:     case 0x202f: /* NARROW NO-BREAK SPACE */
3082:     case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
3083:     case 0x3000: /* IDEOGRAPHIC SPACE */
3084:     break;
3085:     }
3086: }
3087: break;
3088:
3089: case OP_NOT_VSPACE:
3090: for (i = 1; i <= min; i++)
3091: {
3092:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3093:     GETCHARINC(c, eptr);
3094:     switch(c)
3095:     {
3096:     default: break;
3097:     case 0x0a: /* LF */
3098:     case 0x0b: /* VT */
3099:     case 0x0c: /* FF */
3100:     case 0x0d: /* CR */
3101:     case 0x85: /* NEL */
3102:     case 0x2028: /* LINE SEPARATOR */
3103:     case 0x2029: /* PARAGRAPH SEPARATOR */
3104:         RRETURN(MATCH_NOMATCH);
3105:     }
3106: }
3107: break;
3108:
3109: case OP_VSPACE:
3110: for (i = 1; i <= min; i++)

```

```

3111:     {
3112:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3113:     GETCHARINC(c, eptr);
3114:     switch(c)
3115:     {
3116:     default: RRETURN(MATCH_NOMATCH);
3117:     case 0x0a:    /* LF */
3118:     case 0x0b:    /* VT */
3119:     case 0x0c:    /* FF */
3120:     case 0x0d:    /* CR */
3121:     case 0x85:    /* NEL */
3122:     case 0x2028:  /* LINE SEPARATOR */
3123:     case 0x2029:  /* PARAGRAPH SEPARATOR */
3124:     break;
3125:     }
3126:     }
3127:     break;
3128:
3129:     case OP_NOT_DIGIT:
3130:     for (i = 1; i <= min; i++)
3131:     {
3132:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3133:     GETCHARINC(c, eptr);
3134:     if (c < 128 && (md->ctypes[c] & ctype_digit) != 0)
3135:     RRETURN(MATCH_NOMATCH);
3136:     }
3137:     break;
3138:
3139:     case OP_DIGIT:
3140:     for (i = 1; i <= min; i++)
3141:     {
3142:     if (eptr >= md->end_subject ||
3143:     *eptr >= 128 || (md->ctypes[*eptr++] & ctype_digit) == 0)
3144:     RRETURN(MATCH_NOMATCH);
3145:     /* No need to skip more bytes - we know it's a 1-byte character */
3146:     }
3147:     break;
3148:
3149:     case OP_NOT_WHITESPACE:
3150:     for (i = 1; i <= min; i++)
3151:     {

```

```

3152:     if (eptr >= md->end_subject ||
3153:         (*eptr < 128 && (md->ctypes[*eptr] & ctype_space) != 0))
3154:         RRETURN(MATCH_NOMATCH);
3155:     while (++eptr < md->end_subject && (*eptr & 0xc0) == 0x80);
3156: }
3157: break;
3158:
3159: case OP_WHITESPACE:
3160: for (i = 1; i <= min; i++)
3161: {
3162:     if (eptr >= md->end_subject ||
3163:         *eptr >= 128 || (md->ctypes[*eptr++] & ctype_space) == 0)
3164:         RRETURN(MATCH_NOMATCH);
3165:     /* No need to skip more bytes - we know it's a 1-byte character */
3166: }
3167: break;
3168:
3169: case OP_NOT_WORDCHAR:
3170: for (i = 1; i <= min; i++)
3171: {
3172:     if (eptr >= md->end_subject ||
3173:         (*eptr < 128 && (md->ctypes[*eptr] & ctype_word) != 0))
3174:         RRETURN(MATCH_NOMATCH);
3175:     while (++eptr < md->end_subject && (*eptr & 0xc0) == 0x80);
3176: }
3177: break;
3178:
3179: case OP_WORDCHAR:
3180: for (i = 1; i <= min; i++)
3181: {
3182:     if (eptr >= md->end_subject ||
3183:         *eptr >= 128 || (md->ctypes[*eptr++] & ctype_word) == 0)
3184:         RRETURN(MATCH_NOMATCH);
3185:     /* No need to skip more bytes - we know it's a 1-byte character */
3186: }
3187: break;
3188:
3189: default:
3190: RRETURN(PCRE_ERROR_INTERNAL);
3191: } /* End switch(ctype) */
3192:

```

```

3193:     else
3194: #endif    /* SUPPORT_UTF8 */
3195:
3196:     /* Code for the non-UTF-8 case for minimum matching of operators other
3197:     than OP_PROP and OP_NOTPROP. We can assume that there are the minimum
3198:     number of bytes present, as this was tested above. */
3199:
3200:     switch(ctype)
3201:     {
3202:     case OP_ANY:
3203:         for (i = 1; i <= min; i++)
3204:         {
3205:             if (IS_NEWLINE(eptr)) RRETURN(MATCH_NOMATCH);
3206:             eptr++;
3207:         }
3208:         break;
3209:
3210:     case OP_ALLANY:
3211:         eptr += min;
3212:         break;
3213:
3214:     case OP_ANYBYTE:
3215:         eptr += min;
3216:         break;
3217:
3218:     /* Because of the CRLF case, we can't assume the minimum number of
3219:     bytes are present in this case. */
3220:
3221:     case OP_ANYNL:
3222:         for (i = 1; i <= min; i++)
3223:         {
3224:             if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3225:             switch(*eptr++)
3226:             {
3227:             default: RRETURN(MATCH_NOMATCH);
3228:             case 0x000d:
3229:                 if (eptr < md->end_subject && *eptr == 0x000a) eptr++;
3230:                 break;
3231:             case 0x000a:
3232:                 break;
3233:

```

```

3234:     case 0x000b:
3235:     case 0x000c:
3236:     case 0x0085:
3237:         if (md->bsr_anycrlf) RRETURN(MATCH_NOMATCH);
3238:         break;
3239:     }
3240: }
3241: break;
3242:
3243: case OP_NOT_HSPACE:
3244: for (i = 1; i <= min; i++)
3245: {
3246:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3247:     switch(*eptr++)
3248:     {
3249:         default: break;
3250:         case 0x09:    /* HT */
3251:         case 0x20:    /* SPACE */
3252:         case 0xa0:    /* NBSP */
3253:             RRETURN(MATCH_NOMATCH);
3254:     }
3255: }
3256: break;
3257:
3258: case OP_HSPACE:
3259: for (i = 1; i <= min; i++)
3260: {
3261:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3262:     switch(*eptr++)
3263:     {
3264:         default: RRETURN(MATCH_NOMATCH);
3265:         case 0x09:    /* HT */
3266:         case 0x20:    /* SPACE */
3267:         case 0xa0:    /* NBSP */
3268:             break;
3269:     }
3270: }
3271: break;
3272:
3273: case OP_NOT_VSPACE:
3274: for (i = 1; i <= min; i++)

```



```

3275:     {
3276:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3277:     switch(*eptr++)
3278:     {
3279:     default: break;
3280:     case 0x0a:    /* LF */
3281:     case 0x0b:    /* VT */
3282:     case 0x0c:    /* FF */
3283:     case 0x0d:    /* CR */
3284:     case 0x85:    /* NEL */
3285:     RRETURN(MATCH_NOMATCH);
3286:     }
3287:     }
3288:     break;
3289:
3290:     case OP_VSPACE:
3291:     for (i = 1; i <= min; i++)
3292:     {
3293:     if (eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3294:     switch(*eptr++)
3295:     {
3296:     default: RRETURN(MATCH_NOMATCH);
3297:     case 0x0a:    /* LF */
3298:     case 0x0b:    /* VT */
3299:     case 0x0c:    /* FF */
3300:     case 0x0d:    /* CR */
3301:     case 0x85:    /* NEL */
3302:     break;
3303:     }
3304:     }
3305:     break;
3306:
3307:     case OP_NOT_DIGIT:
3308:     for (i = 1; i <= min; i++)
3309:     if ((md->ctypes[*eptr++] & ctype_digit) != 0) RRETURN(MATCH_NOMATCH);
3310:     break;
3311:
3312:     case OP_DIGIT:
3313:     for (i = 1; i <= min; i++)
3314:     if ((md->ctypes[*eptr++] & ctype_digit) == 0) RRETURN(MATCH_NOMATCH);
3315:     break;

```

```

3316:
3317:     case OP_NOT_WHITESPACE:
3318:         for (i = 1; i <= min; i++)
3319:             if ((md->ctypes[*eptr++] & ctype_space) != 0) RRETURN(MATCH_NOMATCH);
3320:         break;
3321:
3322:     case OP_WHITESPACE:
3323:         for (i = 1; i <= min; i++)
3324:             if ((md->ctypes[*eptr++] & ctype_space) == 0) RRETURN(MATCH_NOMATCH);
3325:         break;
3326:
3327:     case OP_NOT_WORDCHAR:
3328:         for (i = 1; i <= min; i++)
3329:             if ((md->ctypes[*eptr++] & ctype_word) != 0)
3330:                 RRETURN(MATCH_NOMATCH);
3331:         break;
3332:
3333:     case OP_WORDCHAR:
3334:         for (i = 1; i <= min; i++)
3335:             if ((md->ctypes[*eptr++] & ctype_word) == 0)
3336:                 RRETURN(MATCH_NOMATCH);
3337:         break;
3338:
3339:     default:
3340:         RRETURN(PCRE_ERROR_INTERNAL);
3341:     }
3342: }
3343:
3344: /* If min = max, continue at the same level without recursing */
3345:
3346: if (min == max) continue;
3347:
3348: /* If minimizing, we have to test the rest of the pattern before each
3349: subsequent match. Again, separate the UTF-8 case for speed, and also
3350: separate the UCP cases. */
3351:
3352: if (minimize)
3353: {
3354: #ifdef SUPPORT_UCP
3355:     if (prop_type >= 0)
3356:     {

```

```

3357: switch(prop_type)
3358: {
3359:     case PT_ANY:
3360:         for (fi = min;; fi++)
3361:         {
3362:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM36);
3363:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3364:             if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3365:             GETCHARINC(c, eptr);
3366:             if (prop_fail_result) RRETURN(MATCH_NOMATCH);
3367:         }
3368:         /* Control never gets here */
3369:
3370:     case PT_LAMP:
3371:         for (fi = min;; fi++)
3372:         {
3373:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM37);
3374:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3375:             if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3376:             GETCHARINC(c, eptr);
3377:             prop_chartype = UCD_CHARTYPE(c);
3378:             if ((prop_chartype == ucp_Lu ||
3379:                 prop_chartype == ucp_Ll ||
3380:                 prop_chartype == ucp_Lt) == prop_fail_result)
3381:                 RRETURN(MATCH_NOMATCH);
3382:         }
3383:         /* Control never gets here */
3384:
3385:     case PT_GC:
3386:         for (fi = min;; fi++)
3387:         {
3388:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM38);
3389:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3390:             if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3391:             GETCHARINC(c, eptr);
3392:             prop_category = UCD_CATEGORY(c);
3393:             if ((prop_category == prop_value) == prop_fail_result)
3394:                 RRETURN(MATCH_NOMATCH);
3395:         }
3396:         /* Control never gets here */
3397:

```

```

3398:     case PT_PC:
3399:     for (fi = min;; fi++)
3400:     {
3401:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM39);
3402:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3403:         if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3404:         GETCHARINC(c, eptr);
3405:         prop_chartype = UCD_CHARTYPE(c);
3406:         if ((prop_chartype == prop_value) == prop_fail_result)
3407:             RRETURN(MATCH_NOMATCH);
3408:     }
3409:     /* Control never gets here */
3410:
3411:     case PT_SC:
3412:     for (fi = min;; fi++)
3413:     {
3414:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM40);
3415:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3416:         if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);
3417:         GETCHARINC(c, eptr);
3418:         prop_script = UCD_SCRIPT(c);
3419:         if ((prop_script == prop_value) == prop_fail_result)
3420:             RRETURN(MATCH_NOMATCH);
3421:     }
3422:     /* Control never gets here */
3423:
3424:     default:
3425:         RRETURN(PCRE_ERROR_INTERNAL);
3426:     }
3427: }
3428:
3429: /* Match extended Unicode sequences. We will get here only if the
3430: support is in the binary; otherwise a compile-time error occurs. */
3431:
3432: else if (ctype == OP_EXTUNI)
3433: {
3434:     for (fi = min;; fi++)
3435:     {
3436:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM41);
3437:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3438:         if (fi >= max || eptr >= md->end_subject) RRETURN(MATCH_NOMATCH);

```

```

3439:    GETCHARINCTEST(c, eptr);
3440:    prop_category = UCD_CATEGORY(c);
3441:    if (prop_category == ucp_M) RRETURN(MATCH_NOMATCH);
3442:    while (eptr < md->end_subject)
3443:    {
3444:        int len = 1;
3445:        if (!utf8) c = *eptr; else
3446:        {
3447:            GETCHARLEN(c, eptr, len);
3448:        }
3449:        prop_category = UCD_CATEGORY(c);
3450:        if (prop_category != ucp_M) break;
3451:        eptr += len;
3452:    }
3453: }
3454: }
3455:
3456:     else
3457: #endif    /* SUPPORT_UCP */
3458:
3459: #ifdef SUPPORT_UTF8
3460:     /* UTF-8 mode */
3461:     if (utf8)
3462:     {
3463:         for (fi = min;; fi++)
3464:         {
3465:             RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM42);
3466:             if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3467:             if (fi >= max || eptr >= md->end_subject ||
3468:                 (ctype == OP_ANY && IS_NEWLINE(eptr)))
3469:                 RRETURN(MATCH_NOMATCH);
3470:
3471:             GETCHARINC(c, eptr);
3472:             switch(ctype)
3473:             {
3474:                 case OP_ANY:    /* This is the non-NL case */
3475:                 case OP_ALLANY:
3476:                 case OP_ANYBYTE:
3477:                     break;
3478:
3479:                 case OP_ANYNL:

```

```

3480:     switch(c)
3481:     {
3482:         default: RRETURN(MATCH_NOMATCH);
3483:         case 0x000d:
3484:             if (eptr < md->end_subject && *eptr == 0x0a) eptr++;
3485:             break;
3486:         case 0x000a:
3487:             break;
3488:
3489:         case 0x000b:
3490:         case 0x000c:
3491:         case 0x0085:
3492:         case 0x2028:
3493:         case 0x2029:
3494:             if (md->bsr_anycrlf) RRETURN(MATCH_NOMATCH);
3495:             break;
3496:     }
3497:     break;
3498:
3499:     case OP_NOT_HSPACE:
3500:     switch(c)
3501:     {
3502:         default: break;
3503:         case 0x09:    /* HT */
3504:         case 0x20:    /* SPACE */
3505:         case 0xa0:    /* NBSP */
3506:         case 0x1680:  /* OGHAM SPACE MARK */
3507:         case 0x180e:  /* MONGOLIAN VOWEL SEPARATOR */
3508:         case 0x2000:  /* EN QUAD */
3509:         case 0x2001:  /* EM QUAD */
3510:         case 0x2002:  /* EN SPACE */
3511:         case 0x2003:  /* EM SPACE */
3512:         case 0x2004:  /* THREE-PER-EM SPACE */
3513:         case 0x2005:  /* FOUR-PER-EM SPACE */
3514:         case 0x2006:  /* SIX-PER-EM SPACE */
3515:         case 0x2007:  /* FIGURE SPACE */
3516:         case 0x2008:  /* PUNCTUATION SPACE */
3517:         case 0x2009:  /* THIN SPACE */
3518:         case 0x200A:  /* HAIR SPACE */
3519:         case 0x202f:  /* NARROW NO-BREAK SPACE */
3520:         case 0x205f:  /* MEDIUM MATHEMATICAL SPACE */

```

```

3521:     case 0x3000: /* IDEOGRAPHIC SPACE */
3522:         RRETURN(MATCH_NOMATCH);
3523:     }
3524:     break;
3525:
3526:     case OP_HSPACE:
3527:         switch(c)
3528:         {
3529:             default: RRETURN(MATCH_NOMATCH);
3530:             case 0x09: /* HT */
3531:             case 0x20: /* SPACE */
3532:             case 0xa0: /* NBSP */
3533:             case 0x1680: /* OGHAM SPACE MARK */
3534:             case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
3535:             case 0x2000: /* EN QUAD */
3536:             case 0x2001: /* EM QUAD */
3537:             case 0x2002: /* EN SPACE */
3538:             case 0x2003: /* EM SPACE */
3539:             case 0x2004: /* THREE-PER-EM SPACE */
3540:             case 0x2005: /* FOUR-PER-EM SPACE */
3541:             case 0x2006: /* SIX-PER-EM SPACE */
3542:             case 0x2007: /* FIGURE SPACE */
3543:             case 0x2008: /* PUNCTUATION SPACE */
3544:             case 0x2009: /* THIN SPACE */
3545:             case 0x200A: /* HAIR SPACE */
3546:             case 0x202f: /* NARROW NO-BREAK SPACE */
3547:             case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
3548:             case 0x3000: /* IDEOGRAPHIC SPACE */
3549:             break;
3550:         }
3551:         break;
3552:
3553:     case OP_NOT_VSPACE:
3554:         switch(c)
3555:         {
3556:             default: break;
3557:             case 0x0a: /* LF */
3558:             case 0x0b: /* VT */
3559:             case 0x0c: /* FF */
3560:             case 0x0d: /* CR */
3561:             case 0x85: /* NEL */

```

```

3562:     case 0x2028: /* LINE SEPARATOR */
3563:     case 0x2029: /* PARAGRAPH SEPARATOR */
3564:         RRETURN(MATCH_NOMATCH);
3565:     }
3566:     break;
3567:
3568:     case OP_VSPACE:
3569:     switch(c)
3570:     {
3571:         default: RRETURN(MATCH_NOMATCH);
3572:         case 0x0a: /* LF */
3573:         case 0x0b: /* VT */
3574:         case 0x0c: /* FF */
3575:         case 0x0d: /* CR */
3576:         case 0x85: /* NEL */
3577:         case 0x2028: /* LINE SEPARATOR */
3578:         case 0x2029: /* PARAGRAPH SEPARATOR */
3579:             break;
3580:     }
3581:     break;
3582:
3583:     case OP_NOT_DIGIT:
3584:     if (c < 256 && (md->ctypes[c] & ctype_digit) != 0)
3585:         RRETURN(MATCH_NOMATCH);
3586:     break;
3587:
3588:     case OP_DIGIT:
3589:     if (c >= 256 || (md->ctypes[c] & ctype_digit) == 0)
3590:         RRETURN(MATCH_NOMATCH);
3591:     break;
3592:
3593:     case OP_NOT_WHITESPACE:
3594:     if (c < 256 && (md->ctypes[c] & ctype_space) != 0)
3595:         RRETURN(MATCH_NOMATCH);
3596:     break;
3597:
3598:     case OP_WHITESPACE:
3599:     if (c >= 256 || (md->ctypes[c] & ctype_space) == 0)
3600:         RRETURN(MATCH_NOMATCH);
3601:     break;
3602:

```



```

3603:     case OP_NOT_WORDCHAR:
3604:         if (c < 256 && (md->ctypes[c] & ctype_word) != 0)
3605:             RRETURN(MATCH_NOMATCH);
3606:         break;
3607:
3608:     case OP_WORDCHAR:
3609:         if (c >= 256 || (md->ctypes[c] & ctype_word) == 0)
3610:             RRETURN(MATCH_NOMATCH);
3611:         break;
3612:
3613:     default:
3614:         RRETURN(PCRE_ERROR_INTERNAL);
3615:     }
3616: }
3617: }
3618: else
3619: #endif
3620: /* Not UTF-8 mode */
3621: {
3622:     for (fi = min;; fi++)
3623:     {
3624:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM43);
3625:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3626:         if (fi >= max || eptr >= md->end_subject ||
3627:             (ctype == OP_ANY && IS_NEWLINE(eptr)))
3628:             RRETURN(MATCH_NOMATCH);
3629:
3630:         c = *eptr++;
3631:         switch(ctype)
3632:         {
3633:             case OP_ANY: /* This is the non-NL case */
3634:             case OP_ALLANY:
3635:             case OP_ANYBYTE:
3636:                 break;
3637:
3638:             case OP_ANYNL:
3639:                 switch(c)
3640:                 {
3641:                     default: RRETURN(MATCH_NOMATCH);
3642:                     case 0x000d:
3643:                         if (eptr < md->end_subject && *eptr == 0x0a) eptr++;

```

```

3644:     break;
3645:
3646:     case 0x000a:
3647:     break;
3648:
3649:     case 0x000b:
3650:     case 0x000c:
3651:     case 0x0085:
3652:     if (md->bsr_anycrlf) RRETURN(MATCH_NOMATCH);
3653:     break;
3654:     }
3655: break;
3656:
3657: case OP_NOT_HSPACE:
3658: switch(c)
3659: {
3660:     default: break;
3661:     case 0x09:    /* HT */
3662:     case 0x20:    /* SPACE */
3663:     case 0xa0:    /* NBSP */
3664:     RRETURN(MATCH_NOMATCH);
3665: }
3666: break;
3667:
3668: case OP_HSPACE:
3669: switch(c)
3670: {
3671:     default: RRETURN(MATCH_NOMATCH);
3672:     case 0x09:    /* HT */
3673:     case 0x20:    /* SPACE */
3674:     case 0xa0:    /* NBSP */
3675:     break;
3676: }
3677: break;
3678:
3679: case OP_NOT_VSPACE:
3680: switch(c)
3681: {
3682:     default: break;
3683:     case 0x0a:    /* LF */
3684:     case 0x0b:    /* VT */

```

```

3685:     case 0x0c:    /* FF */
3686:     case 0x0d:    /* CR */
3687:     case 0x85:    /* NEL */
3688:         RRETURN(MATCH_NOMATCH);
3689:     }
3690:     break;
3691:
3692:     case OP_VSPACE:
3693:     switch(c)
3694:     {
3695:         default: RRETURN(MATCH_NOMATCH);
3696:         case 0x0a:    /* LF */
3697:         case 0x0b:    /* VT */
3698:         case 0x0c:    /* FF */
3699:         case 0x0d:    /* CR */
3700:         case 0x85:    /* NEL */
3701:             break;
3702:     }
3703:     break;
3704:
3705:     case OP_NOT_DIGIT:
3706:     if ((md->ctypes[c] & ctype_digit) != 0) RRETURN(MATCH_NOMATCH);
3707:     break;
3708:
3709:     case OP_DIGIT:
3710:     if ((md->ctypes[c] & ctype_digit) == 0) RRETURN(MATCH_NOMATCH);
3711:     break;
3712:
3713:     case OP_NOT_WHITESPACE:
3714:     if ((md->ctypes[c] & ctype_space) != 0) RRETURN(MATCH_NOMATCH);
3715:     break;
3716:
3717:     case OP_WHITESPACE:
3718:     if ((md->ctypes[c] & ctype_space) == 0) RRETURN(MATCH_NOMATCH);
3719:     break;
3720:
3721:     case OP_NOT_WORDCHAR:
3722:     if ((md->ctypes[c] & ctype_word) != 0) RRETURN(MATCH_NOMATCH);
3723:     break;
3724:
3725:     case OP_WORDCHAR:

```

```

3726:     if ((md->ctypes[c] & ctype_word) == 0) RRETURN(MATCH_NOMATCH);
3727:     break;
3728:
3729:     default:
3730:         RRETURN(PCRE_ERROR_INTERNAL);
3731:     }
3732: }
3733: }
3734: /* Control never gets here */
3735: }
3736:
3737: /* If maximizing, it is worth using inline code for speed, doing the type
3738: test once at the start (i.e. keep it out of the loop). Again, keep the
3739: UTF-8 and UCP stuff separate. */
3740:
3741: else
3742: {
3743:     pp = eptr; /* Remember where we started */
3744:
3745: #ifdef SUPPORT_UCP
3746:     if (prop_type >= 0)
3747:     {
3748:         switch(prop_type)
3749:         {
3750:             case PT_ANY:
3751:                 for (i = min; i < max; i++)
3752:                 {
3753:                     int len = 1;
3754:                     if (eptr >= md->end_subject) break;
3755:                     GETCHARLEN(c, eptr, len);
3756:                     if (prop_fail_result) break;
3757:                     eptr += len;
3758:                 }
3759:                 break;
3760:
3761:             case PT_LAMP:
3762:                 for (i = min; i < max; i++)
3763:                 {
3764:                     int len = 1;
3765:                     if (eptr >= md->end_subject) break;
3766:                     GETCHARLEN(c, eptr, len);

```

```

3767:     prop_chartype = UCD_CHARTYPE(c);
3768:     if ((prop_chartype == ucp_Lu ||
3769:         prop_chartype == ucp_Ll ||
3770:         prop_chartype == ucp_Lt) == prop_fail_result)
3771:         break;
3772:     eptr += len;
3773: }
3774: break;
3775:
3776: case PT_GC:
3777: for (i = min; i < max; i++)
3778: {
3779:     int len = 1;
3780:     if (eptr >= md->end_subject) break;
3781:     GETCHARLEN(c, eptr, len);
3782:     prop_category = UCD_CATEGORY(c);
3783:     if ((prop_category == prop_value) == prop_fail_result)
3784:         break;
3785:     eptr += len;
3786: }
3787: break;
3788:
3789: case PT_PC:
3790: for (i = min; i < max; i++)
3791: {
3792:     int len = 1;
3793:     if (eptr >= md->end_subject) break;
3794:     GETCHARLEN(c, eptr, len);
3795:     prop_chartype = UCD_CHARTYPE(c);
3796:     if ((prop_chartype == prop_value) == prop_fail_result)
3797:         break;
3798:     eptr += len;
3799: }
3800: break;
3801:
3802: case PT_SC:
3803: for (i = min; i < max; i++)
3804: {
3805:     int len = 1;
3806:     if (eptr >= md->end_subject) break;
3807:     GETCHARLEN(c, eptr, len);

```

```

3808:     prop_script = UCD_SCRIPT(c);
3809:     if ((prop_script == prop_value) == prop_fail_result)
3810:         break;
3811:     eptr += len;
3812: }
3813: break;
3814: }
3815:
3816: /* eptr is now past the end of the maximum run */
3817:
3818: if (possessive) continue;
3819: for(;;)
3820: {
3821:     RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM44);
3822:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3823:     if (eptr-- == pp) break;    /* Stop if tried at original pos */
3824:     if (utf8) BACKCHAR(eptr);
3825: }
3826: }
3827:
3828: /* Match extended Unicode sequences. We will get here only if the
3829: support is in the binary; otherwise a compile-time error occurs. */
3830:
3831: else if (ctype == OP_EXTUNI)
3832: {
3833:     for (i = min; i < max; i++)
3834:     {
3835:         if (eptr >= md->end_subject) break;
3836:         GETCHARINCTEST(c, eptr);
3837:         prop_category = UCD_CATEGORY(c);
3838:         if (prop_category == ucp_M) break;
3839:         while (eptr < md->end_subject)
3840:         {
3841:             int len = 1;
3842:             if (!utf8) c = *eptr; else
3843:             {
3844:                 GETCHARLEN(c, eptr, len);
3845:             }
3846:             prop_category = UCD_CATEGORY(c);
3847:             if (prop_category != ucp_M) break;
3848:             eptr += len;

```

```

3849:     }
3850: }
3851:
3852: /* eptr is now past the end of the maximum run */
3853:
3854: if (possessive) continue;
3855: for(;;)
3856: {
3857:     RMATCH(eptr, encode, offset_top, md, ims, eptrb, 0, RM45);
3858:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);
3859:     if (eptr-- == pp) break;    /* Stop if tried at original pos */
3860:     for (;;)                  /* Move back over one extended */
3861:     {
3862:         int len = 1;
3863:         if (!utf8) c = *eptr; else
3864:         {
3865:             BACKCHAR(eptr);
3866:             GETCHARLEN(c, eptr, len);
3867:         }
3868:         prop_category = UCD_CATEGORY(c);
3869:         if (prop_category != ucp_M) break;
3870:         eptr--;
3871:     }
3872: }
3873: }
3874:
3875: else
3876: #endif /* SUPPORT_UCP */
3877:
3878: #ifdef SUPPORT_UTF8
3879:     /* UTF-8 mode */
3880:
3881:     if (utf8)
3882:     {
3883:         switch(ctype)
3884:         {
3885:             case OP_ANY:
3886:                 if (max < INT_MAX)
3887:                 {
3888:                     for (i = min; i < max; i++)
3889:                     {

```

```

3890:     if (eptr >= md->end_subject || IS_NEWLINE(eptr)) break;
3891:     eptr++;
3892:     while (eptr < md->end_subject && (*eptr & 0xc0) == 0x80) eptr++;
3893: }
3894: }
3895:
3896: /* Handle unlimited UTF-8 repeat */
3897:
3898: else
3899: {
3900:     for (i = min; i < max; i++)
3901:     {
3902:         if (eptr >= md->end_subject || IS_NEWLINE(eptr)) break;
3903:         eptr++;
3904:         while (eptr < md->end_subject && (*eptr & 0xc0) == 0x80) eptr++;
3905:     }
3906: }
3907: break;
3908:
3909: case OP_ALLANY:
3910: if (max < INT_MAX)
3911: {
3912:     for (i = min; i < max; i++)
3913:     {
3914:         if (eptr >= md->end_subject) break;
3915:         eptr++;
3916:         while (eptr < md->end_subject && (*eptr & 0xc0) == 0x80) eptr++;
3917:     }
3918: }
3919: else eptr = md->end_subject; /* Unlimited UTF-8 repeat */
3920: break;
3921:
3922: /* The byte case is the same as non-UTF8 */
3923:
3924: case OP_ANYBYTE:
3925: c = max - min;
3926: if (c > (unsigned int)(md->end_subject - eptr))
3927:     c = md->end_subject - eptr;
3928: eptr += c;
3929: break;
3930:

```



```

3931:     case OP_ANYNL:
3932:     for (i = min; i < max; i++)
3933:     {
3934:         int len = 1;
3935:         if (eptr >= md->end_subject) break;
3936:         GETCHARLEN(c, eptr, len);
3937:         if (c == 0x000d)
3938:         {
3939:             if (++eptr >= md->end_subject) break;
3940:             if (*eptr == 0x000a) eptr++;
3941:         }
3942:     else
3943:     {
3944:         if (c != 0x000a &&
3945:             (md->bsr_anycrlf ||
3946:              (c != 0x000b && c != 0x000c &&
3947:               c != 0x0085 && c != 0x2028 && c != 0x2029))))
3948:             break;
3949:         eptr += len;
3950:     }
3951: }
3952: break;
3953:
3954: case OP_NOT_HSPACE:
3955: case OP_HSPACE:
3956: for (i = min; i < max; i++)
3957: {
3958:     BOOL gotspace;
3959:     int len = 1;
3960:     if (eptr >= md->end_subject) break;
3961:     GETCHARLEN(c, eptr, len);
3962:     switch(c)
3963:     {
3964:         default: gotspace = FALSE; break;
3965:         case 0x09: /* HT */
3966:         case 0x20: /* SPACE */
3967:         case 0xa0: /* NBSP */
3968:         case 0x1680: /* OGHAM SPACE MARK */
3969:         case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
3970:         case 0x2000: /* EN QUAD */
3971:         case 0x2001: /* EM QUAD */

```

```

3972:     case 0x2002: /* EN SPACE */
3973:     case 0x2003: /* EM SPACE */
3974:     case 0x2004: /* THREE-PER-EM SPACE */
3975:     case 0x2005: /* FOUR-PER-EM SPACE */
3976:     case 0x2006: /* SIX-PER-EM SPACE */
3977:     case 0x2007: /* FIGURE SPACE */
3978:     case 0x2008: /* PUNCTUATION SPACE */
3979:     case 0x2009: /* THIN SPACE */
3980:     case 0x200A: /* HAIR SPACE */
3981:     case 0x202f: /* NARROW NO-BREAK SPACE */
3982:     case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
3983:     case 0x3000: /* IDEOGRAPHIC SPACE */
3984:     gotospace = TRUE;
3985:     break;
3986: }
3987: if (gotospace == (ctype == OP_NOT_HSPACE)) break;
3988: eptr += len;
3989: }
3990: break;
3991:
3992: case OP_NOT_VSPACE:
3993: case OP_VSPACE:
3994: for (i = min; i < max; i++)
3995: {
3996:     BOOL gotospace;
3997:     int len = 1;
3998:     if (eptr >= md->end_subject) break;
3999:     GETCHARLEN(c, eptr, len);
4000:     switch(c)
4001:     {
4002:     default: gotospace = FALSE; break;
4003:     case 0x0a: /* LF */
4004:     case 0x0b: /* VT */
4005:     case 0x0c: /* FF */
4006:     case 0x0d: /* CR */
4007:     case 0x85: /* NEL */
4008:     case 0x2028: /* LINE SEPARATOR */
4009:     case 0x2029: /* PARAGRAPH SEPARATOR */
4010:     gotospace = TRUE;
4011:     break;
4012: }

```

```

4013:     if (gotspace == (ctype == OP_NOT_VSPACE)) break;
4014:     eptr += len;
4015: }
4016: break;
4017:
4018: case OP_NOT_DIGIT:
4019: for (i = min; i < max; i++)
4020: {
4021:     int len = 1;
4022:     if (eptr >= md->end_subject) break;
4023:     GETCHARLEN(c, eptr, len);
4024:     if (c < 256 && (md->ctypes[c] & ctype_digit) != 0) break;
4025:     eptr += len;
4026: }
4027: break;
4028:
4029: case OP_DIGIT:
4030: for (i = min; i < max; i++)
4031: {
4032:     int len = 1;
4033:     if (eptr >= md->end_subject) break;
4034:     GETCHARLEN(c, eptr, len);
4035:     if (c >= 256 || (md->ctypes[c] & ctype_digit) == 0) break;
4036:     eptr += len;
4037: }
4038: break;
4039:
4040: case OP_NOT_WHITESPACE:
4041: for (i = min; i < max; i++)
4042: {
4043:     int len = 1;
4044:     if (eptr >= md->end_subject) break;
4045:     GETCHARLEN(c, eptr, len);
4046:     if (c < 256 && (md->ctypes[c] & ctype_space) != 0) break;
4047:     eptr += len;
4048: }
4049: break;
4050:
4051: case OP_WHITESPACE:
4052: for (i = min; i < max; i++)
4053: {

```

```

4054:     int len = 1;
4055:     if (eptr >= md->end_subject) break;
4056:     GETCHARLEN(c, eptr, len);
4057:     if (c >= 256 || (md->ctypes[c] & ctype_space) == 0) break;
4058:     eptr += len;
4059: }
4060: break;
4061:
4062: case OP_NOT_WORDCHAR:
4063: for (i = min; i < max; i++)
4064: {
4065:     int len = 1;
4066:     if (eptr >= md->end_subject) break;
4067:     GETCHARLEN(c, eptr, len);
4068:     if (c < 256 && (md->ctypes[c] & ctype_word) != 0) break;
4069:     eptr += len;
4070: }
4071: break;
4072:
4073: case OP_WORDCHAR:
4074: for (i = min; i < max; i++)
4075: {
4076:     int len = 1;
4077:     if (eptr >= md->end_subject) break;
4078:     GETCHARLEN(c, eptr, len);
4079:     if (c >= 256 || (md->ctypes[c] & ctype_word) == 0) break;
4080:     eptr += len;
4081: }
4082: break;
4083:
4084: default:
4085:     RRETURN(PCRE_ERROR_INTERNAL);
4086: }
4087:
4088: /* eptr is now past the end of the maximum run */
4089:
4090: if (possessive) continue;
4091: for(;;)
4092: {
4093:     RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM46);
4094:     if (rrc != MATCH_NOMATCH) RRETURN(rrc);

```

```

4095:     if (eptr-- == pp) break;    /* Stop if tried at original pos */
4096:     BACKCHAR(eptr);
4097: }
4098: }
4099: else
4100: #endif /* SUPPORT_UTF8 */
4101:
4102: /* Not UTF-8 mode */
4103: {
4104:     switch(ctype)
4105:     {
4106:     case OP_ANY:
4107:         for (i = min; i < max; i++)
4108:         {
4109:             if (eptr >= md->end_subject || IS_NEWLINE(eptr)) break;
4110:             eptr++;
4111:         }
4112:         break;
4113:
4114:     case OP_ALLANY:
4115:     case OP_ANYBYTE:
4116:         c = max - min;
4117:         if (c > (unsigned int)(md->end_subject - eptr))
4118:             c = md->end_subject - eptr;
4119:         eptr += c;
4120:         break;
4121:
4122:     case OP_ANYNL:
4123:         for (i = min; i < max; i++)
4124:         {
4125:             if (eptr >= md->end_subject) break;
4126:             c = *eptr;
4127:             if (c == 0x000d)
4128:             {
4129:                 if (++eptr >= md->end_subject) break;
4130:                 if (*eptr == 0x000a) eptr++;
4131:             }
4132:         }
4133:     else
4134:     {
4135:         if (c != 0x000a &&
            (md->bsr_anycrlf ||

```

```

4136:         (c != 0x000b && c != 0x000c && c != 0x0085)))
4137:     break;
4138:     eptr++;
4139: }
4140: }
4141: break;
4142:
4143: case OP_NOT_HSPACE:
4144: for (i = min; i < max; i++)
4145: {
4146:     if (eptr >= md->end_subject) break;
4147:     c = *eptr;
4148:     if (c == 0x09 || c == 0x20 || c == 0xa0) break;
4149:     eptr++;
4150: }
4151: break;
4152:
4153: case OP_HSPACE:
4154: for (i = min; i < max; i++)
4155: {
4156:     if (eptr >= md->end_subject) break;
4157:     c = *eptr;
4158:     if (c != 0x09 && c != 0x20 && c != 0xa0) break;
4159:     eptr++;
4160: }
4161: break;
4162:
4163: case OP_NOT_VSPACE:
4164: for (i = min; i < max; i++)
4165: {
4166:     if (eptr >= md->end_subject) break;
4167:     c = *eptr;
4168:     if (c == 0x0a || c == 0x0b || c == 0x0c || c == 0x0d || c == 0x85)
4169:         break;
4170:     eptr++;
4171: }
4172: break;
4173:
4174: case OP_VSPACE:
4175: for (i = min; i < max; i++)
4176: {

```

```

4177:     if (eptr >= md->end_subject) break;
4178:     c = *eptr;
4179:     if (c != 0x0a && c != 0x0b && c != 0x0c && c != 0x0d && c != 0x85)
4180:         break;
4181:     eptr++;
4182: }
4183: break;
4184:
4185: case OP_NOT_DIGIT:
4186: for (i = min; i < max; i++)
4187: {
4188:     if (eptr >= md->end_subject || (md->ctypes[*eptr] & ctype_digit) != 0)
4189:         break;
4190:     eptr++;
4191: }
4192: break;
4193:
4194: case OP_DIGIT:
4195: for (i = min; i < max; i++)
4196: {
4197:     if (eptr >= md->end_subject || (md->ctypes[*eptr] & ctype_digit) == 0)
4198:         break;
4199:     eptr++;
4200: }
4201: break;
4202:
4203: case OP_NOT_WHITESPACE:
4204: for (i = min; i < max; i++)
4205: {
4206:     if (eptr >= md->end_subject || (md->ctypes[*eptr] & ctype_space) != 0)
4207:         break;
4208:     eptr++;
4209: }
4210: break;
4211:
4212: case OP_WHITESPACE:
4213: for (i = min; i < max; i++)
4214: {
4215:     if (eptr >= md->end_subject || (md->ctypes[*eptr] & ctype_space) == 0)
4216:         break;
4217:     eptr++;

```

```

4218:     }
4219:     break;
4220:
4221:     case OP_NOT_WORDCHAR:
4222:     for (i = min; i < max; i++)
4223:     {
4224:         if (eptr >= md->end_subject || (md->ctypes[*eptr] & ctype_word) != 0)
4225:             break;
4226:         eptr++;
4227:     }
4228:     break;
4229:
4230:     case OP_WORDCHAR:
4231:     for (i = min; i < max; i++)
4232:     {
4233:         if (eptr >= md->end_subject || (md->ctypes[*eptr] & ctype_word) == 0)
4234:             break;
4235:         eptr++;
4236:     }
4237:     break;
4238:
4239:     default:
4240:     RRETURN(PCRE_ERROR_INTERNAL);
4241:     }
4242:
4243:     /* eptr is now past the end of the maximum run */
4244:
4245:     if (possessive) continue;
4246:     while (eptr >= pp)
4247:     {
4248:         RMATCH(eptr, ecode, offset_top, md, ims, eptrb, 0, RM47);
4249:         eptr--;
4250:         if (rrc != MATCH_NOMATCH) RRETURN(rrc);
4251:     }
4252:     }
4253:
4254:     /* Get here if we can't make it match with any permitted repetitions */
4255:
4256:     RRETURN(MATCH_NOMATCH);
4257:     }
4258:     /* Control never gets here */

```



```

4259:
4260:  /* There's been some horrible disaster. Arrival here can only mean there is
4261:  something seriously wrong in the code above or the OP_xxx definitions. */
4262:
4263:  default:
4264:  DPRINTF(("Unknown opcode %d \n", *ecode));
4265:  RRETURN(PCRE_ERROR_UNKNOWN_OPCODE);
4266:  }
4267:
4268:  /* Do not stick any code in here without much thought; it is assumed
4269:  that "continue" in the code above comes out to here to repeat the main
4270:  loop. */
4271:
4272:  }          /* End of main loop */
4273: /* Control never reaches here */
4274:
4275:
4276: /* When compiling to use the heap rather than the stack for recursive calls to
4277: match(), the RRETURN() macro jumps here. The number that is saved in
4278: frame->Xwhere indicates which label we actually want to return to. */
4279:
4280: #ifdef NO_RECURSE
4281: #define LBL(val) case val: goto L_RM##val;
4282: HEAP_RETURN:
4283: switch (frame->Xwhere)
4284: {
4285: LBL( 1) LBL( 2) LBL( 3) LBL( 4) LBL( 5) LBL( 6) LBL( 7) LBL( 8)
4286: LBL( 9) LBL(10) LBL(11) LBL(12) LBL(13) LBL(14) LBL(15) LBL(17)
4287: LBL(19) LBL(24) LBL(25) LBL(26) LBL(27) LBL(29) LBL(31) LBL(33)
4288: LBL(35) LBL(43) LBL(47) LBL(48) LBL(49) LBL(50) LBL(51) LBL(52)
4289: LBL(53) LBL(54)
4290: #ifdef SUPPORT_UTF8
4291: LBL(16) LBL(18) LBL(20) LBL(21) LBL(22) LBL(23) LBL(28) LBL(30)
4292: LBL(32) LBL(34) LBL(42) LBL(46)
4293: #ifdef SUPPORT_UCP
4294: LBL(36) LBL(37) LBL(38) LBL(39) LBL(40) LBL(41) LBL(44) LBL(45)
4295: #endif /* SUPPORT_UCP */
4296: #endif /* SUPPORT_UTF8 */
4297: default:
4298: DPRINTF(("jump error in pcre match: label %d non-existent \n", frame->Xwhere));
4299: return PCRE_ERROR_INTERNAL;

```

```

4300: }
4301: #undef LBL
4302: #endif /* NO_RECURSE */
4303: }
4304:
4305:
4306: /*****
4307: *****/
4308:          RECURSION IN THE match() FUNCTION
4309:
4310: Undefine all the macros that were defined above to handle this. */
4311:
4312: #ifdef NO_RECURSE
4313: #undef eptr
4314: #undef ecode
4315: #undef mstart
4316: #undef offset_top
4317: #undef ims
4318: #undef eptrb
4319: #undef flags
4320:
4321: #undef callpat
4322: #undef charptr
4323: #undef data
4324: #undef next
4325: #undef pp
4326: #undef prev
4327: #undef saved_eptr
4328:
4329: #undef new_recursive
4330:
4331: #undef cur_is_word
4332: #undef condition
4333: #undef prev_is_word
4334:
4335: #undef original_ims
4336:
4337: #undef ctype
4338: #undef length
4339: #undef max
4340: #undef min

```

```

4341: #undef number
4342: #undef offset
4343: #undef op
4344: #undef save_capture_last
4345: #undef save_offset1
4346: #undef save_offset2
4347: #undef save_offset3
4348: #undef stacksave
4349:
4350: #undef newptrb
4351:
4352: #endif
4353:
4354: /* These two are defined as macros in both cases */
4355:
4356: #undef fc
4357: #undef fi
4358:
4359: /*****
4360: *****/
4361:
4362:
4363:
4364: /*****
4365: *      Execute a Regular Expression      *
4366: *****/
4367:
4368: /* This function applies a compiled re to a subject string and picks out
4369: portions of the string if it matches. Two elements in the vector are set for
4370: each substring: the offsets to the start and end of the substring.
4371:
4372: Arguments:
4373: argument_re    points to the compiled expression
4374: extra_data     points to extra data or is NULL
4375: subject        points to the subject string
4376: length         length of subject string (may contain binary zeros)
4377: start_offset   where to start in the subject string
4378: options        option bits
4379: offsets        points to a vector of ints to be filled in with offsets
4380: offsetcount    the number of elements in the vector
4381:

```

```

4382: Returns:      > 0 => success; value is the number of elements filled in
4383:               = 0 => success, but offsets is not big enough
4384:               -1 => failed to match
4385:               < -1 => some kind of unexpected problem
4386: */
4387:
4388: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
4389: pcre_exec(const pcre *argument_re, const pcre_extra *extra_data,
4390:  PCRE_SPTR subject, int length, int start_offset, int options, int *offsets,
4391:  int offsetcount)
4392: {
4393: int rc, resetcount, ocount;
4394: int first_byte = -1;
4395: int req_byte = -1;
4396: int req_byte2 = -1;
4397: int newline;
4398: unsigned long int ims;
4399: BOOL using_temporary_offsets = FALSE;
4400: BOOL anchored;
4401: BOOL startline;
4402: BOOL firstline;
4403: BOOL first_byte_caseless = FALSE;
4404: BOOL req_byte_caseless = FALSE;
4405: BOOL utf8;
4406: match_data match_block;
4407: match_data *md = &match_block;
4408: const uschar *tables;
4409: const uschar *start_bits = NULL;
4410: USPTR start_match = (USPTR)subject + start_offset;
4411: USPTR end_subject;
4412: USPTR req_byte_ptr = start_match - 1;
4413:
4414: pcre_study_data internal_study;
4415: const pcre_study_data *study;
4416:
4417: real_pcre internal_re;
4418: const real_pcre *external_re = (const real_pcre *)argument_re;
4419: const real_pcre *re = external_re;
4420:
4421: /* Plausibility checks */
4422:

```

```

4423: if ((options & ~PUBLIC_EXEC_OPTIONS) != 0) return PCRE_ERROR_BADOPTION;
4424: if (re == NULL || subject == NULL ||
4425:  (offsets == NULL && offsetcount > 0)) return PCRE_ERROR_NULL;
4426: if (offsetcount < 0) return PCRE_ERROR_BADCOUNT;
4427:
4428: /* Fish out the optional data from the extra_data structure, first setting
4429:  the default values. */
4430:
4431: study = NULL;
4432: md->match_limit = MATCH_LIMIT;
4433: md->match_limit_recursion = MATCH_LIMIT_RECURSION;
4434: md->callout_data = NULL;
4435:
4436: /* The table pointer is always in native byte order. */
4437:
4438: tables = external_re->tables;
4439:
4440: if (extra_data != NULL)
4441: {
4442:  register unsigned int flags = extra_data->flags;
4443:  if ((flags & PCRE_EXTRA_STUDY_DATA) != 0)
4444:    study = (const pcre_study_data *)extra_data->study_data;
4445:  if ((flags & PCRE_EXTRA_MATCH_LIMIT) != 0)
4446:    md->match_limit = extra_data->match_limit;
4447:  if ((flags & PCRE_EXTRA_MATCH_LIMIT_RECURSION) != 0)
4448:    md->match_limit_recursion = extra_data->match_limit_recursion;
4449:  if ((flags & PCRE_EXTRA_CALLOUT_DATA) != 0)
4450:    md->callout_data = extra_data->callout_data;
4451:  if ((flags & PCRE_EXTRA_TABLES) != 0) tables = extra_data->tables;
4452: }
4453:
4454: /* If the exec call supplied NULL for tables, use the inbuilt ones. This
4455:  is a feature that makes it possible to save compiled regex and re-use them
4456:  in other programs later. */
4457:
4458: if (tables == NULL) tables = _pcre_default_tables;
4459:
4460: /* Check that the first field in the block is the magic number. If it is not,
4461:  test for a regex that was compiled on a host of opposite endianness. If this is
4462:  the case, flipped values are put in internal_re and internal_study if there was
4463:  study data too. */

```

```

4464:
4465: if (re->magic_number != MAGIC_NUMBER)
4466: {
4467: re = _pcre_try_flipped(re, &internal_re, study, &internal_study);
4468: if (re == NULL) return PCRE_ERROR_BADMAGIC;
4469: if (study != NULL) study = &internal_study;
4470: }
4471:
4472: /* Set up other data */
4473:
4474: anchored = ((re->options | options) & PCRE_ANCHORED) != 0;
4475: startline = (re->flags & PCRE_STARTLINE) != 0;
4476: firstline = (re->options & PCRE_FIRSTLINE) != 0;
4477:
4478: /* The code starts after the real_pcre block and the capture name table. */
4479:
4480: md->start_code = (const uschar *)external_re + re->name_table_offset +
4481: re->name_count * re->name_entry_size;
4482:
4483: md->start_subject = (USPTR)subject;
4484: md->start_offset = start_offset;
4485: md->end_subject = md->start_subject + length;
4486: end_subject = md->end_subject;
4487:
4488: md->endonly = (re->options & PCRE_DOLLAR_ENDONLY) != 0;
4489: utf8 = md->utf8 = (re->options & PCRE_UTF8) != 0;
4490: md->jscript_compat = (re->options & PCRE_JAVASCRIPT_COMPAT) != 0;
4491:
4492: md->notbol = (options & PCRE_NOTBOL) != 0;
4493: md->noteol = (options & PCRE_NOTEOL) != 0;
4494: md->notempty = (options & PCRE_NOTEMPTY) != 0;
4495: md->partial = (options & PCRE_PARTIAL) != 0;
4496: md->hitend = FALSE;
4497:
4498: md->recursive = NULL;          /* No recursion at top level */
4499:
4500: md->lcc = tables + lcc_offset;
4501: md->ctypes = tables + ctypes_offset;
4502:
4503: /* Handle different \R options. */
4504:

```

```

4505: switch (options & (PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE))
4506: {
4507: case 0:
4508: if ((re->options & (PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE)) != 0)
4509:   md->bsr_anycrlf = (re->options & PCRE_BSR_ANYCRLF) != 0;
4510: else
4511: #ifdef BSR_ANYCRLF
4512:   md->bsr_anycrlf = TRUE;
4513: #else
4514:   md->bsr_anycrlf = FALSE;
4515: #endif
4516: break;
4517:
4518: case PCRE_BSR_ANYCRLF:
4519:   md->bsr_anycrlf = TRUE;
4520:   break;
4521:
4522: case PCRE_BSR_UNICODE:
4523:   md->bsr_anycrlf = FALSE;
4524:   break;
4525:
4526: default: return PCRE_ERROR_BADNEWLINE;
4527: }
4528:
4529: /* Handle different types of newline. The three bits give eight cases. If
4530: nothing is set at run time, whatever was used at compile time applies. */
4531:
4532: switch (((options & PCRE_NEWLINE_BITS) == 0)? re->options :
4533:   (pcre_uint32)options) & PCRE_NEWLINE_BITS)
4534: {
4535: case 0: newline = NEWLINE; break; /* Compile-time default */
4536: case PCRE_NEWLINE_CR: newline = '\r'; break;
4537: case PCRE_NEWLINE_LF: newline = '\n'; break;
4538: case PCRE_NEWLINE_CR+
4539:   PCRE_NEWLINE_LF: newline = ('\r' << 8) | '\n'; break;
4540: case PCRE_NEWLINE_ANY: newline = -1; break;
4541: case PCRE_NEWLINE_ANYCRLF: newline = -2; break;
4542: default: return PCRE_ERROR_BADNEWLINE;
4543: }
4544:
4545: if (newline == -2)

```

```

4546: {
4547: md->nltype = NLTYPE_ANYCRLF;
4548: }
4549: else if (newline < 0)
4550: {
4551: md->nltype = NLTYPE_ANY;
4552: }
4553: else
4554: {
4555: md->nltype = NLTYPE_FIXED;
4556: if (newline > 255)
4557: {
4558: md->nllen = 2;
4559: md->nl[0] = (newline >> 8) & 255;
4560: md->nl[1] = newline & 255;
4561: }
4562: else
4563: {
4564: md->nllen = 1;
4565: md->nl[0] = newline;
4566: }
4567: }
4568:
4569: /* Partial matching is supported only for a restricted set of regexes at the
4570: moment. */
4571:
4572: if (md->partial && (re->flags & PCRE_NOPARTIAL) != 0)
4573: return PCRE_ERROR_BADPARTIAL;
4574:
4575: /* Check a UTF-8 string if required. Unfortunately there's no way of passing
4576: back the character offset. */
4577:
4578: #ifdef SUPPORT_UTF8
4579: if (utf8 && (options & PCRE_NO_UTF8_CHECK) == 0)
4580: {
4581: if (_pcre_valid_utf8((uschar *)subject, length) >= 0)
4582: return PCRE_ERROR_BADUTF8;
4583: if (start_offset > 0 && start_offset < length)
4584: {
4585: int tb = ((uschar *)subject)[start_offset];
4586: if (tb > 127)

```



```

4587:  {
4588:    tb &= 0xc0;
4589:    if (tb != 0 && tb != 0xc0) return PCRE_ERROR_BADUTF8_OFFSET;
4590:  }
4591: }
4592: }
4593: #endif
4594:
4595: /* The ims options can vary during the matching as a result of the presence
4596: of (?ims) items in the pattern. They are kept in a local variable so that
4597: restoring at the exit of a group is easy. */
4598:
4599: ims = re->options & (PCRE_CASELESS|PCRE_MULTILINE|PCRE_DOTALL);
4600:
4601: /* If the expression has got more back references than the offsets supplied can
4602: hold, we get a temporary chunk of working store to use during the matching.
4603: Otherwise, we can use the vector supplied, rounding down its size to a multiple
4604: of 3. */
4605:
4606: ocount = offsetcount - (offsetcount % 3);
4607:
4608: if (re->top_backref > 0 && re->top_backref >= ocount/3)
4609: {
4610:   ocount = re->top_backref * 3 + 3;
4611:   md->offset_vector = (int *)(pcre_malloc)(ocount * sizeof(int));
4612:   if (md->offset_vector == NULL) return PCRE_ERROR_NOMEMORY;
4613:   using_temporary_offsets = TRUE;
4614:   DPRINTF(("Got memory to hold back references \n"));
4615: }
4616: else md->offset_vector = offsets;
4617:
4618: md->offset_end = ocount;
4619: md->offset_max = (2*ocount)/3;
4620: md->offset_overflow = FALSE;
4621: md->capture_last = -1;
4622:
4623: /* Compute the minimum number of offsets that we need to reset each time. Doing
4624: this makes a huge difference to execution time when there aren't many brackets
4625: in the pattern. */
4626:
4627: resetcount = 2 + re->top_bracket * 2;

```

```

4628: if (resetcount > offsetcount) resetcount = ocount;
4629:
4630: /* Reset the working variable associated with each extraction. These should
4631: never be used unless previously set, but they get saved and restored, and so we
4632: initialize them to avoid reading uninitialized locations. */
4633:
4634: if (md->offset_vector != NULL)
4635: {
4636:     register int *iptr = md->offset_vector + ocount;
4637:     register int *iend = iptr - resetcount/2 + 1;
4638:     while (--iptr >= iend) *iptr = -1;
4639: }
4640:
4641: /* Set up the first character to match, if available. The first_byte value is
4642: never set for an anchored regular expression, but the anchoring may be forced
4643: at run time, so we have to test for anchoring. The first char may be unset for
4644: an unanchored pattern, of course. If there's no first char and the pattern was
4645: studied, there may be a bitmap of possible first characters. */
4646:
4647: if (!anchored)
4648: {
4649:     if ((re->flags & PCRE_FIRSTSET) != 0)
4650:     {
4651:         first_byte = re->first_byte & 255;
4652:         if ((first_byte_caseless = ((re->first_byte & REQ_CASELESS) != 0)) == TRUE)
4653:             first_byte = md->lcc[first_byte];
4654:     }
4655:     else
4656:         if (!startline && study != NULL &&
4657:             (study->options & PCRE_STUDY_MAPPED) != 0)
4658:             start_bits = study->start_bits;
4659: }
4660:
4661: /* For anchored or unanchored matches, there may be a "last known required
4662: character" set. */
4663:
4664: if ((re->flags & PCRE_REQCHSET) != 0)
4665: {
4666:     req_byte = re->req_byte & 255;
4667:     req_byte_caseless = (re->req_byte & REQ_CASELESS) != 0;
4668:     req_byte2 = (tables + fcc_offset)[req_byte]; /* case flipped */

```

```

4669: }
4670:
4671:
4672:                                     /*
===== */
4673:
4674: /* Loop for handling unanchored repeated matching attempts; for anchored regexs
4675: the loop runs just once. */
4676:
4677: for(;;)
4678: {
4679:     USPTR save_end_subject = end_subject;
4680:     USPTR new_start_match;
4681:
4682:     /* Reset the maximum number of extractions we might see. */
4683:
4684:     if (md->offset_vector != NULL)
4685:     {
4686:         register int *iptr = md->offset_vector;
4687:         register int *iend = iptr + resetcount;
4688:         while (iptr < iend) *iptr++ = -1;
4689:     }
4690:
4691:     /* Advance to a unique first char if possible. If firstline is TRUE, the
4692:     start of the match is constrained to the first line of a multiline string.
4693:     That is, the match must be before or at the first newline. Implement this by
4694:     temporarily adjusting end_subject so that we stop scanning at a newline. If
4695:     the match fails at the newline, later code breaks this loop. */
4696:
4697:     if (firstline)
4698:     {
4699:         USPTR t = start_match;
4700: #ifdef SUPPORT_UTF8
4701:         if (utf8)
4702:         {
4703:             while (t < md->end_subject && !IS_NEWLINE(t))
4704:             {
4705:                 t++;
4706:                 while (t < end_subject && (*t & 0xc0) == 0x80) t++;
4707:             }
4708:         }

```

```

4709:  else
4710: #endif
4711:  while (t < md->end_subject && !IS_NEWLINE(t)) t++;
4712:  end_subject = t;
4713:  }
4714:
4715:  /* Now advance to a unique first byte if there is one. */
4716:
4717:  if (first_byte >= 0)
4718:  {
4719:    if (first_byte_caseless)
4720:      while (start_match < end_subject && md->lcc[*start_match] != first_byte)
4721:        start_match++;
4722:    else
4723:      while (start_match < end_subject && *start_match != first_byte)
4724:        start_match++;
4725:  }
4726:
4727:  /* Or to just after a linebreak for a multiline match */
4728:
4729:  else if (startline)
4730:  {
4731:    if (start_match > md->start_subject + start_offset)
4732:    {
4733: #ifdef SUPPORT_UTF8
4734:    if (utf8)
4735:    {
4736:      while (start_match < end_subject && !WAS_NEWLINE(start_match))
4737:      {
4738:        start_match++;
4739:        while(start_match < end_subject && (*start_match & 0xc0) == 0x80)
4740:          start_match++;
4741:      }
4742:    }
4743:    else
4744: #endif
4745:      while (start_match < end_subject && !WAS_NEWLINE(start_match))
4746:        start_match++;
4747:
4748:    /* If we have just passed a CR and the newline option is ANY or ANYCRLF,
4749:       and we are now at a LF, advance the match position by one more character.

```

```

4750:  */
4751:
4752:  if (start_match[-1] == '\r' &&
4753:      (md->nlttype == NLTYPE_ANY || md->nlttype == NLTYPE_ANYCRLF) &&
4754:      start_match < end_subject &&
4755:      *start_match == '\n')
4756:      start_match++;
4757:  }
4758: }
4759:
4760: /* Or to a non-unique first byte after study */
4761:
4762: else if (start_bits != NULL)
4763: {
4764:  while (start_match < end_subject)
4765:  {
4766:      register unsigned int c = *start_match;
4767:      if (((start_bits[c/8] & (1 << (c&7))) == 0) start_match++;
4768:      else break;
4769:  }
4770: }
4771:
4772: /* Restore fudged end_subject */
4773:
4774: end_subject = save_end_subject;
4775:
4776: #ifdef DEBUG /* Sigh. Some compilers never learn. */
4777:  printf(">>>> Match against: ");
4778:  pchars(start_match, end_subject - start_match, TRUE, md);
4779:  printf("\n");
4780: #endif
4781:
4782: /* If req_byte is set, we know that that character must appear in the subject
4783:  for the match to succeed. If the first character is set, req_byte must be
4784:  later in the subject; otherwise the test starts at the match point. This
4785:  optimization can save a huge amount of backtracking in patterns with nested
4786:  unlimited repeats that aren't going to match. Writing separate code for
4787:  cased/caseless versions makes it go faster, as does using an autoincrement
4788:  and backing off on a match.
4789:
4790:  HOWEVER: when the subject string is very, very long, searching to its end can

```

```

4791: take a long time, and give bad performance on quite ordinary patterns. This
4792: showed up when somebody was matching something like /^ \d+C/ on a 32-megabyte
4793: string... so we don't do this when the string is sufficiently long.
4794:
4795: ALSO: this processing is disabled when partial matching is requested.
4796: */
4797:
4798: if (req_byte >= 0 &&
4799:     end_subject - start_match < REQ_BYTE_MAX &&
4800:     !md->partial)
4801: {
4802:     register USPTR p = start_match + ((first_byte >= 0)? 1 : 0);
4803:
4804:     /* We don't need to repeat the search if we haven't yet reached the
4805:        place we found it at last time. */
4806:
4807:     if (p > req_byte_ptr)
4808:     {
4809:         if (req_byte_caseless)
4810:         {
4811:             while (p < end_subject)
4812:             {
4813:                 register int pp = *p++;
4814:                 if (pp == req_byte || pp == req_byte2) { p--; break; }
4815:             }
4816:         }
4817:         else
4818:         {
4819:             while (p < end_subject)
4820:             {
4821:                 if (*p++ == req_byte) { p--; break; }
4822:             }
4823:         }
4824:
4825:         /* If we can't find the required character, break the matching loop,
4826:            forcing a match failure. */
4827:
4828:         if (p >= end_subject)
4829:         {
4830:             rc = MATCH_NOMATCH;
4831:             break;

```

```

4832:     }
4833:
4834:     /* If we have found the required character, save the point where we
4835:        found it, so that we don't search again next time round the loop if
4836:        the start hasn't passed this character yet. */
4837:
4838:     req_byte_ptr = p;
4839: }
4840: }
4841:
4842: /* OK, we can now run the match. */
4843:
4844: md->start_match_ptr = start_match;
4845: md->match_call_count = 0;
4846: rc = match(start_match, md->start_code, start_match, 2, md, ims, NULL, 0, 0);
4847:
4848: switch(rc)
4849: {
4850:     /* NOMATCH and PRUNE advance by one character. THEN at this level acts
4851:        exactly like PRUNE. */
4852:
4853:     case MATCH_NOMATCH:
4854:     case MATCH_PRUNE:
4855:     case MATCH_THEN:
4856:         new_start_match = start_match + 1;
4857: #ifdef SUPPORT_UTF8
4858:         if (utf8)
4859:             while(new_start_match < end_subject && (*new_start_match & 0xc0) == 0x80)
4860:                 new_start_match++;
4861: #endif
4862:         break;
4863:
4864:     /* SKIP passes back the next starting point explicitly. */
4865:
4866:     case MATCH_SKIP:
4867:         new_start_match = md->start_match_ptr;
4868:         break;
4869:
4870:     /* COMMIT disables the bumpalong, but otherwise behaves as NOMATCH. */
4871:
4872:     case MATCH_COMMIT:

```

```

4873: rc = MATCH_NOMATCH;
4874: goto ENDLOOP;
4875:
4876: /* Any other return is some kind of error. */
4877:
4878: default:
4879: goto ENDLOOP;
4880: }
4881:
4882: /* Control reaches here for the various types of "no match at this point"
4883: result. Reset the code to MATCH_NOMATCH for subsequent checking. */
4884:
4885: rc = MATCH_NOMATCH;
4886:
4887: /* If PCRE_FIRSTLINE is set, the match must happen before or at the first
4888: newline in the subject (though it may continue over the newline). Therefore,
4889: if we have just failed to match, starting at a newline, do not continue. */
4890:
4891: if (firstline && IS_NEWLINE(start_match)) break;
4892:
4893: /* Advance to new matching position */
4894:
4895: start_match = new_start_match;
4896:
4897: /* Break the loop if the pattern is anchored or if we have passed the end of
4898: the subject. */
4899:
4900: if (anchored || start_match > end_subject) break;
4901:
4902: /* If we have just passed a CR and we are now at a LF, and the pattern does
4903: not contain any explicit matches for \r or \n, and the newline option is CRLF
4904: or ANY or ANYCRLF, advance the match position by one more character. */
4905:
4906: if (start_match[-1] == '\r' &&
4907:     start_match < end_subject &&
4908:     *start_match == '\n' &&
4909:     (re->flags & PCRE_HASCORLRF) == 0 &&
4910:     (md->nltypes == NLTYPE_ANY ||
4911:      md->nltypes == NLTYPE_ANYCRLF ||
4912:      md->nllen == 2))
4913:     start_match++;

```



```

4914:
4915: } /* End of for(;;) "bumpalong" loop */
4916:
4917:                                     /*
=====*/
4918:
4919: /* We reach here when rc is not MATCH_NOMATCH, or if one of the stopping
4920: conditions is true:
4921:
4922: (1) The pattern is anchored or the match was failed by (*COMMIT);
4923:
4924: (2) We are past the end of the subject;
4925:
4926: (3) PCRE_FIRSTLINE is set and we have failed to match at a newline, because
4927:     this option requests that a match occur at or before the first newline in
4928:     the subject.
4929:
4930: When we have a match and the offset vector is big enough to deal with any
4931: backreferences, captured substring offsets will already be set up. In the case
4932: where we had to get some local store to hold offsets for backreference
4933: processing, copy those that we can. In this case there need not be overflow if
4934: certain parts of the pattern were not used, even though there are more
4935: capturing parentheses than vector slots. */
4936:
4937: ENDLOOP:
4938:
4939: if (rc == MATCH_MATCH)
4940: {
4941:   if (using_temporary_offsets)
4942:   {
4943:     if (offsetcount >= 4)
4944:     {
4945:       memcpy(offsets + 2, md->offset_vector + 2,
4946:         (offsetcount - 2) * sizeof(int));
4947:       DPRINTF(("Copied offsets from temporary memory \n"));
4948:     }
4949:     if (md->end_offset_top > offsetcount) md->offset_overflow = TRUE;
4950:     DPRINTF(("Freeing temporary memory \n"));
4951:     (pcre_free)(md->offset_vector);
4952:   }
4953:

```

```

4954: /* Set the return code to the number of captured strings, or 0 if there are
4955: too many to fit into the vector. */
4956:
4957: rc = md->offset_overflow? 0 : md->end_offset_top/2;
4958:
4959: /* If there is space, set up the whole thing as substring 0. The value of
4960: md->start_match_ptr might be modified if \K was encountered on the success
4961: matching path. */
4962:
4963: if (offsetcount < 2) rc = 0; else
4964: {
4965:   offsets[0] = md->start_match_ptr - md->start_subject;
4966:   offsets[1] = md->end_match_ptr - md->start_subject;
4967: }
4968:
4969: DPRINTF((">>>> returning %d \n", rc));
4970: return rc;
4971: }
4972:
4973: /* Control gets here if there has been an error, or if the overall match
4974: attempt has failed at all permitted starting positions. */
4975:
4976: if (using_temporary_offsets)
4977: {
4978:   DPRINTF(("Freeing temporary memory \n"));
4979:   (pcre_free)(md->offset_vector);
4980: }
4981:
4982: if (rc != MATCH_NOMATCH)
4983: {
4984:   DPRINTF((">>>> error: returning %d \n", rc));
4985:   return rc;
4986: }
4987: else if (md->partial && md->hitend)
4988: {
4989:   DPRINTF((">>>> returning PCRE_ERROR_PARTIAL \n"));
4990:   return PCRE_ERROR_PARTIAL;
4991: }
4992: else
4993: {
4994:   DPRINTF((">>>> returning PCRE_ERROR_NOMATCH \n"));

```

```
4995: return PCRE_ERROR_NOMATCH;
4996: }
4997: }
4998:
4999: /* End of pcre_exec.c */
```

File: sdm/VxWorks/libRegex/pcre.h

```
1: /*****
2: *   Perl-Compatible Regular Expressions   *
3: *****/
4:
5: /* This is the public header file for the PCRE library, to be #included by
6: applications that call the PCRE functions.
7:
8:      Copyright (c) 1997-2008 University of Cambridge
9:
10: -----
11: Redistribution and use in source and binary forms, with or without
12: modification, are permitted provided that the following conditions are met:
13:
14: * Redistributions of source code must retain the above copyright notice,
15:   this list of conditions and the following disclaimer.
16:
17: * Redistributions in binary form must reproduce the above copyright
18:   notice, this list of conditions and the following disclaimer in the
19:   documentation and/or other materials provided with the distribution.
20:
21: * Neither the name of the University of Cambridge nor the names of its
22:   contributors may be used to endorse or promote products derived from
23:   this software without specific prior written permission.
24:
25: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
26: "AS IS"
27: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
28: THE
29: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
30: PURPOSE
31: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
32: BE
33: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
34: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
35: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
36: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
37: IN
38: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
39: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
40: POSSIBILITY OF SUCH DAMAGE.
```

```

36: -----
37: */
38:
39: #ifndef _PCRE_H
40: #define _PCRE_H
41:
42: /* The current PCRE version information. */
43:
44: #define PCRE_MAJOR      7
45: #define PCRE_MINOR      8
46: #define PCRE_PRERELEASE
47: #define PCRE_DATE       2008-09-05
48:
49: /* When an application links to a PCRE DLL in Windows, the symbols that are
50: imported have to be identified as such. When building PCRE, the appropriate
51: export setting is defined in pcre_internal.h, which includes this file. So we
52: don't change existing definitions of PCRE_EXP_DECL and PCRECPP_EXP_DECL. */
53:
54: #if defined(_WIN32) && !defined(PCRE_STATIC)
55: #  ifndef PCRE_EXP_DECL
56: #    define PCRE_EXP_DECL extern __declspec(dllimport)
57: #  endif
58: #  ifdef __cplusplus
59: #    ifndef PCRECPP_EXP_DECL
60: #      define PCRECPP_EXP_DECL extern __declspec(dllimport)
61: #    endif
62: #    ifndef PCRECPP_EXP_DEFN
63: #      define PCRECPP_EXP_DEFN __declspec(dllimport)
64: #    endif
65: #  endif
66: #endif
67:
68: /* By default, we use the standard "extern" declarations. */
69:
70: #ifndef PCRE_EXP_DECL
71: #  ifdef __cplusplus
72: #    define PCRE_EXP_DECL extern "C"
73: #  else
74: #    define PCRE_EXP_DECL extern
75: #  endif
76: #endif

```

```

77:
78: #ifdef __cplusplus
79: # ifndef PCRECPP_EXP_DECL
80: #   define PCRECPP_EXP_DECL extern
81: # endif
82: # ifndef PCRECPP_EXP_DEFN
83: #   define PCRECPP_EXP_DEFN
84: # endif
85: #endif
86:
87: /* Have to include stdlib.h in order to ensure that size_t is defined;
88: it is needed here for malloc. */
89:
90: #include <stdlib.h>
91:
92: /* Allow for C++ users */
93:
94: #ifdef __cplusplus
95: extern "C" {
96: #endif
97:
98: /* Options */
99:
100: #define PCRE_CASELESS      0x00000001
101: #define PCRE_MULTILINE     0x00000002
102: #define PCRE_DOTALL       0x00000004
103: #define PCRE_EXTENDED      0x00000008
104: #define PCRE_ANCHORED     0x00000010
105: #define PCRE_DOLLAR_ENDONLY 0x00000020
106: #define PCRE_EXTRA         0x00000040
107: #define PCRE_NOTBOL        0x00000080
108: #define PCRE_NOTEOL        0x00000100
109: #define PCRE_UNGREEDY      0x00000200
110: #define PCRE_NOTEMPTY      0x00000400
111: #define PCRE_UTF8          0x00000800
112: #define PCRE_NO_AUTO_CAPTURE 0x00001000
113: #define PCRE_NO_UTF8_CHECK 0x00002000
114: #define PCRE_AUTO_CALLOUT  0x00004000
115: #define PCRE_PARTIAL       0x00008000
116: #define PCRE_DFA_SHORTEST  0x00010000
117: #define PCRE_DFA_RESTART   0x00020000

```

```

118: #define PCRE_FIRSTLINE      0x00040000
119: #define PCRE_DUPNAMES       0x00080000
120: #define PCRE_NEWLINE_CR     0x00100000
121: #define PCRE_NEWLINE_LF     0x00200000
122: #define PCRE_NEWLINE_CRLF   0x00300000
123: #define PCRE_NEWLINE_ANY    0x00400000
124: #define PCRE_NEWLINE_ANYCRLF 0x00500000
125: #define PCRE_BSR_ANYCRLF    0x00800000
126: #define PCRE_BSR_UNICODE    0x01000000
127: #define PCRE_JAVASCRIPT_COMPAT 0x02000000
128:
129: /* Exec-time and get/set-time error codes */
130:
131: #define PCRE_ERROR_NOMATCH    (-1)
132: #define PCRE_ERROR_NULL      (-2)
133: #define PCRE_ERROR_BADOPTION (-3)
134: #define PCRE_ERROR_BADMAGIC  (-4)
135: #define PCRE_ERROR_UNKNOWN_OPCODE (-5)
136: #define PCRE_ERROR_UNKNOWN_NODE (-5) /* For backward compatibility */
137: #define PCRE_ERROR_NOMEMORY   (-6)
138: #define PCRE_ERROR_NOSUBSTRING (-7)
139: #define PCRE_ERROR_MATCHLIMIT (-8)
140: #define PCRE_ERROR_CALLOUT    (-9) /* Never used by PCRE itself */
141: #define PCRE_ERROR_BADUTF8     (-10)
142: #define PCRE_ERROR_BADUTF8_OFFSET (-11)
143: #define PCRE_ERROR_PARTIAL    (-12)
144: #define PCRE_ERROR_BADPARTIAL (-13)
145: #define PCRE_ERROR_INTERNAL   (-14)
146: #define PCRE_ERROR_BADCOUNT  (-15)
147: #define PCRE_ERROR_DFA_UITEM  (-16)
148: #define PCRE_ERROR_DFA_UCOND  (-17)
149: #define PCRE_ERROR_DFA_UMLIMIT (-18)
150: #define PCRE_ERROR_DFA_WSSIZE (-19)
151: #define PCRE_ERROR_DFA_RECURSE (-20)
152: #define PCRE_ERROR_RECURSIONLIMIT (-21)
153: #define PCRE_ERROR_NULLWSLIMIT (-22) /* No longer actually used */
154: #define PCRE_ERROR_BADNEWLINE (-23)
155:
156: /* Request types for pcre_fullinfo() */
157:
158: #define PCRE_INFO_OPTIONS      0

```

```

159: #define PCRE_INFO_SIZE          1
160: #define PCRE_INFO_CAPTURECOUNT  2
161: #define PCRE_INFO_BACKREFMAX     3
162: #define PCRE_INFO_FIRSTBYTE      4
163: #define PCRE_INFO_FIRSTCHAR      4 /* For backwards compatibility */
164: #define PCRE_INFO_FIRSTTABLE     5
165: #define PCRE_INFO_LASTLITERAL    6
166: #define PCRE_INFO_NAMEENTRYSIZE  7
167: #define PCRE_INFO_NAMECOUNT     8
168: #define PCRE_INFO_NAMETABLE     9
169: #define PCRE_INFO_STUDYSIZE     10
170: #define PCRE_INFO_DEFAULT_TABLES 11
171: #define PCRE_INFO_OKPARTIAL     12
172: #define PCRE_INFO_JCHANGED      13
173: #define PCRE_INFO_HASCORLRF     14
174:
175: /* Request types for pcre_config(). Do not re-arrange, in order to remain
176: compatible. */
177:
178: #define PCRE_CONFIG_UTF8          0
179: #define PCRE_CONFIG_NEWLINE      1
180: #define PCRE_CONFIG_LINK_SIZE    2
181: #define PCRE_CONFIG_POSIX_MALLOC_THRESHOLD 3
182: #define PCRE_CONFIG_MATCH_LIMIT  4
183: #define PCRE_CONFIG_STACKRECURSE 5
184: #define PCRE_CONFIG_UNICODE_PROPERTIES 6
185: #define PCRE_CONFIG_MATCH_LIMIT_RECURSION 7
186: #define PCRE_CONFIG_BSR          8
187:
188: /* Bit flags for the pcre_extra structure. Do not re-arrange or redefine
189: these bits, just add new ones on the end, in order to remain compatible. */
190:
191: #define PCRE_EXTRA_STUDY_DATA     0x0001
192: #define PCRE_EXTRA_MATCH_LIMIT    0x0002
193: #define PCRE_EXTRA_CALLOUT_DATA   0x0004
194: #define PCRE_EXTRA_TABLES         0x0008
195: #define PCRE_EXTRA_MATCH_LIMIT_RECURSION 0x0010
196:
197: /* Types */
198:
199: struct real_pcre;          /* declaration; the definition is private */

```



```

200: typedef struct real_pcre pcre;
201:
202: /* When PCRE is compiled as a C++ library, the subject pointer type can be
203: replaced with a custom type. For conventional use, the public interface is a
204: const char *. */
205:
206: #ifndef PCRE_SPTR
207: #define PCRE_SPTR const char *
208: #endif
209:
210: /* The structure for passing additional data to pcre_exec(). This is defined in
211: such as way as to be extensible. Always add new fields at the end, in order to
212: remain compatible. */
213:
214: typedef struct pcre_extra {
215:   unsigned long int flags;      /* Bits for which fields are set */
216:   void *study_data;            /* Opaque data from pcre_study() */
217:   unsigned long int match_limit; /* Maximum number of calls to match() */
218:   void *callout_data;          /* Data passed back in callouts */
219:   const unsigned char *tables; /* Pointer to character tables */
220:   unsigned long int match_limit_recursion; /* Max recursive calls to match() */
221: } pcre_extra;
222:
223: /* The structure for passing out data via the pcre_callout_function. We use a
224: structure so that new fields can be added on the end in future versions,
225: without changing the API of the function, thereby allowing old clients to work
226: without modification. */
227:
228: typedef struct pcre_callout_block {
229:   int      version;      /* Identifies version of block */
230:   /* ----- Version 0 ----- */
231:   int      callout_number; /* Number compiled into pattern */
232:   int      *offset_vector; /* The offset vector */
233:   PCRE_SPTR subject;      /* The subject being matched */
234:   int      subject_length; /* The length of the subject */
235:   int      start_match;    /* Offset to start of this match attempt */
236:   int      current_position; /* Where we currently are in the subject */
237:   int      capture_top;    /* Max current capture */
238:   int      capture_last;   /* Most recently closed capture */
239:   void     *callout_data;  /* Data passed in with the call */
240:   /* ----- Added for Version 1 ----- */

```

```

241: int      pattern_position; /* Offset to next item in the pattern */
242: int      next_item_length; /* Length of next item in the pattern */
243: /* ----- */
244: } pcre_callout_block;
245:
246: /* Indirection for store get and free functions. These can be set to
247: alternative malloc/free functions if required. Special ones are used in the
248: non-recursive case for "frames". There is also an optional callout function
249: that is triggered by the (?) regex item. For Virtual Pascal, these definitions
250: have to take another form. */
251:
252: #ifndef VPCOMPAT
253: PCRE_EXP_DECL void (*pcre_malloc)(size_t);
254: PCRE_EXP_DECL void (*pcre_free)(void *);
255: PCRE_EXP_DECL void (*pcre_stack_malloc)(size_t);
256: PCRE_EXP_DECL void (*pcre_stack_free)(void *);
257: PCRE_EXP_DECL int (*pcre_callout)(pcre_callout_block *);
258: #else /* VPCOMPAT */
259: PCRE_EXP_DECL void *pcre_malloc(size_t);
260: PCRE_EXP_DECL void pcre_free(void *);
261: PCRE_EXP_DECL void *pcre_stack_malloc(size_t);
262: PCRE_EXP_DECL void pcre_stack_free(void *);
263: PCRE_EXP_DECL int pcre_callout(pcre_callout_block *);
264: #endif /* VPCOMPAT */
265:
266: /* Exported PCRE functions */
267:
268: PCRE_EXP_DECL pcre *pcre_compile(const char *, int, const char **, int *,
269: const unsigned char *);
270: PCRE_EXP_DECL pcre *pcre_compile2(const char *, int, int *, const char **,
271: int *, const unsigned char *);
272: PCRE_EXP_DECL int pcre_config(int, void *);
273: PCRE_EXP_DECL int pcre_copy_named_substring(const pcre *, const char *,
274: int *, int, const char *, char *, int);
275: PCRE_EXP_DECL int pcre_copy_substring(const char *, int *, int, int, char *,
276: int);
277: PCRE_EXP_DECL int pcre_dfa_exec(const pcre *, const pcre_extra *,
278: const char *, int, int, int *, int *, int *, int);
279: PCRE_EXP_DECL int pcre_exec(const pcre *, const pcre_extra *, PCRE_SPTR,
280: int, int, int, int *, int);
281: PCRE_EXP_DECL void pcre_free_substring(const char *);

```

```

282: PCRE_EXP_DECL void pcre_free_substring_list(const char **);
283: PCRE_EXP_DECL int pcre_fullinfo(const pcre *, const pcre_extra *, int,
284:     void *);
285: PCRE_EXP_DECL int pcre_get_named_substring(const pcre *, const char *,
286:     int *, int, const char *, const char **);
287: PCRE_EXP_DECL int pcre_get_stringnumber(const pcre *, const char *);
288: PCRE_EXP_DECL int pcre_get_stringtable_entries(const pcre *, const char *,
289:     char **, char **);
290: PCRE_EXP_DECL int pcre_get_substring(const char *, int *, int, int,
291:     const char **);
292: PCRE_EXP_DECL int pcre_get_substring_list(const char *, int *, int,
293:     const char ***);
294: PCRE_EXP_DECL int pcre_info(const pcre *, int *, int *);
295: PCRE_EXP_DECL const unsigned char *pcre_maketables(void);
296: PCRE_EXP_DECL int pcre_refcount(pcre *, int);
297: PCRE_EXP_DECL pcre_extra *pcre_study(const pcre *, int, const char **);
298: PCRE_EXP_DECL const char *pcre_version(void);
299:
300: #ifdef __cplusplus
301: } /* extern "C" */
302: #endif
303:
304: #endif /* End of pcre.h */

```

File: sdm/VxWorks/libRegex/pcre_globals.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3: *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains global variables that are exported by the PCRE library.
42: PCRE is thread-clean and doesn't use any global variables in the normal sense.
43: However, it calls memory allocation and freeing functions via the four
44: indirections below, and it can optionally do callouts, using the fifth
45: indirection. These values can be changed by the caller, but are shared between
46: all threads. However, when compiling for Virtual Pascal, things are done
47: differently, and global variables are not used (see pcre.in). */
48:
49: #ifdef HAVE_CONFIG_H
50: #include "config.h"
51: #endif
52:
53: #include "pcre_internal.h"
54:
55: #ifndef VPCOMPAT
56: PCRE_EXP_DATA_DEFN void *(*pcre_malloc)(size_t) = malloc;
57: PCRE_EXP_DATA_DEFN void (*pcre_free)(void *) = free;
58: PCRE_EXP_DATA_DEFN void *(*pcre_stack_malloc)(size_t) = malloc;
59: PCRE_EXP_DATA_DEFN void (*pcre_stack_free)(void *) = free;
60: PCRE_EXP_DATA_DEFN int (*pcre_callout)(pcre_callout_block *) = NULL;
61: #endif
62:
63: /* End of pcre_globals.c */

```

File: sdm/VxWorks/libRegex/pcre_dfa_exec.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_dfa_exec(), which is an
42: alternative matching function that uses a sort of DFA algorithm (not a true
43: FSM). This is NOT Perl- compatible, but it has advantages in certain
44: applications. */
45:
46:
47: #ifdef HAVE_CONFIG_H
48: #include "config.h"
49: #endif
50:
51: #define NLBLOCK md          /* Block containing newline information */
52: #define PSSTART start_subject /* Field containing processed string start */
53: #define PSEND end_subject   /* Field containing processed string end */
54:
55: #include "pcre_internal.h"
56:
57:
58: /* For use to indent debugging output */
59:
60: #define SP "          "
61:
62:
63:
64: /*****
65: *      Code parameters and static tables      *
66: *****/
67:
68: /* These are offsets that are used to turn the OP_TYPESTAR and friends opcodes
69: into others, under special conditions. A gap of 20 between the blocks should be
70: enough. The resulting opcodes don't have to be less than 256 because they are
71: never stored, so we push them well clear of the normal opcodes. */
72:
73: #define OP_PROP_EXTRA    300
74: #define OP_EXTUNI_EXTRA  320
75: #define OP_ANYNL_EXTRA   340
76: #define OP_HSPACE_EXTRA  360

```

```

77: #define OP_VSPACE_EXTRA    380
78:
79:
80: /* This table identifies those opcodes that are followed immediately by a
81: character that is to be tested in some way. This makes is possible to
82: centralize the loading of these characters. In the case of Type * etc, the
83: "character" is the opcode for \D, \d, \S, \s, \W, or \w, which will always be a
84: small value. ***NOTE*** If the start of this table is modified, the two tables
85: that follow must also be modified. */
86:
87: static const uschar coptable[] = {
88: 0, /* End */
89: 0, 0, 0, 0, 0, /* \A, \G, \K, \B, \b */
90: 0, 0, 0, 0, 0, 0, /* \D, \d, \S, \s, \W, \w */
91: 0, 0, 0, /* Any, AllAny, Anybyte */
92: 0, 0, 0, /* NOTPROP, PROP, EXTUNI */
93: 0, 0, 0, 0, 0, /* \R, \H, \h, \V, \v */
94: 0, 0, 0, 0, 0, /* \Z, \z, Opt, ^, $ */
95: 1, /* Char */
96: 1, /* Charnc */
97: 1, /* not */
98: /* Positive single-char repeats */
99: 1, 1, 1, 1, 1, 1, /* *, *?, +, +?, ?, ?? */
100: 3, 3, 3, /* upto, minupto, exact */
101: 1, 1, 1, 3, /* *+, ++, ?+, upto+ */
102: /* Negative single-char repeats - only for chars < 256 */
103: 1, 1, 1, 1, 1, 1, /* NOT *, *?, +, +?, ?, ?? */
104: 3, 3, 3, /* NOT upto, minupto, exact */
105: 1, 1, 1, 3, /* NOT *+, ++, ?+, updo+ */
106: /* Positive type repeats */
107: 1, 1, 1, 1, 1, 1, /* Type *, *?, +, +?, ?, ?? */
108: 3, 3, 3, /* Type upto, minupto, exact */
109: 1, 1, 1, 3, /* Type *+, ++, ?+, upto+ */
110: /* Character class & ref repeats */
111: 0, 0, 0, 0, 0, 0, /* *, *?, +, +?, ?, ?? */
112: 0, 0, /* CRRANGE, CRMINRANGE */
113: 0, /* CLASS */
114: 0, /* NCLASS */
115: 0, /* XCLASS - variable length */
116: 0, /* REF */
117: 0, /* RECURSE */

```



```

118: 0,          /* CALLOUT          */
119: 0,          /* Alt              */
120: 0,          /* Ket              */
121: 0,          /* KetRmax          */
122: 0,          /* KetRmin          */
123: 0,          /* Assert           */
124: 0,          /* Assert not       */
125: 0,          /* Assert behind    */
126: 0,          /* Assert behind not */
127: 0,          /* Reverse          */
128: 0, 0, 0, 0, /* ONCE, BRA, CBRA, COND */
129: 0, 0, 0,    /* SBRA, SCBRA, SCOND */
130: 0,          /* CREF             */
131: 0,          /* RREF             */
132: 0,          /* DEF              */
133: 0, 0,       /* BRAZERO, BRAMINZERO */
134: 0, 0, 0, 0, /* PRUNE, SKIP, THEN, COMMIT */
135: 0, 0, 0     /* FAIL, ACCEPT, SKIPZERO */
136: };
137:
138: /* These 2 tables allow for compact code for testing for \D, \d, \S, \s, \W,
139: and \w */
140:
141: static const uschar toptable1[] = {
142: 0, 0, 0, 0, 0, 0,
143: ctype_digit, ctype_digit,
144: ctype_space, ctype_space,
145: ctype_word,  ctype_word,
146: 0, 0          /* OP_ANY, OP_ALLANY */
147: };
148:
149: static const uschar toptable2[] = {
150: 0, 0, 0, 0, 0, 0,
151: ctype_digit, 0,
152: ctype_space, 0,
153: ctype_word, 0,
154: 1, 1          /* OP_ANY, OP_ALLANY */
155: };
156:
157:
158: /* Structure for holding data about a particular state, which is in effect the

```

```

159: current data for an active path through the match tree. It must consist
160: entirely of ints because the working vector we are passed, and which we put
161: these structures in, is a vector of ints. */
162:
163: typedef struct stateblock {
164:     int offset;          /* Offset to opcode */
165:     int count;           /* Count for repeats */
166:     int ims;             /* ims flag bits */
167:     int data;            /* Some use extra data */
168: } stateblock;
169:
170: #define INTS_PER_STATEBLOCK (sizeof(stateblock)/sizeof(int))
171:
172:
173: #ifdef DEBUG
174: /*****
175:  *      Print character string      *
176: *****/
177:
178: /* Character string printing function for debugging.
179:
180: Arguments:
181: p      points to string
182: length number of bytes
183: f      where to print
184:
185: Returns: nothing
186: */
187:
188: static void
189: pchars(unsigned char *p, int length, FILE *f)
190: {
191:     int c;
192:     while (length-- > 0)
193:     {
194:         if (isprint(c = *(p++)))
195:             fprintf(f, "%c", c);
196:         else
197:             fprintf(f, "\\x%02x", c);
198:     }
199: }

```

```

200: #endif
201:
202:
203:
204: /*****
205:  *   Execute a Regular Expression - DFA engine   *
206:  *****/
207:
208: /* This internal function applies a compiled pattern to a subject string,
209: starting at a given point, using a DFA engine. This function is called from the
210: external one, possibly multiple times if the pattern is not anchored. The
211: function calls itself recursively for some kinds of subpattern.
212:
213: Arguments:
214: md          the match_data block with fixed information
215: this_start_code  the opening bracket of this subexpression's code
216: current_subject  where we currently are in the subject string
217: start_offset    start offset in the subject string
218: offsets         vector to contain the matching string offsets
219: offsetcount     size of same
220: workspace       vector of workspace
221: wscount         size of same
222: ims            the current ims flags
223: rlevel         function call recursion level
224: recursing      regex recursive call level
225:
226: Returns:      > 0 => number of match offset pairs placed in offsets
227:              = 0 => offsets overflowed; longest matches are present
228:              -1 => failed to match
229:              < -1 => some kind of unexpected problem
230:
231: The following macros are used for adding states to the two state vectors (one
232: for the current character, one for the following character). */
233:
234: #define ADD_ACTIVE(x,y) \
235: if (active_count++ < wscount) \
236: { \
237: next_active_state->offset = (x); \
238: next_active_state->count = (y); \
239: next_active_state->ims = ims; \
240: next_active_state++; \

```

```

241:  DPRINTF(("%.sADD_ACTIVE(%d,%d) \n", rlevel*2-2, SP, (x), (y))); \
242:  } \
243:  else return PCRE_ERROR_DFA_WSSIZE
244:
245: #define ADD_ACTIVE_DATA(x,y,z) \
246:  if (active_count++ < wscount) \
247:  { \
248:    next_active_state->offset = (x); \
249:    next_active_state->count = (y); \
250:    next_active_state->ims = ims; \
251:    next_active_state->data = (z); \
252:    next_active_state++; \
253:    DPRINTF(("%.sADD_ACTIVE_DATA(%d,%d,%d) \n", rlevel*2-2, SP, (x), (y), (z))); \
254:  } \
255:  else return PCRE_ERROR_DFA_WSSIZE
256:
257: #define ADD_NEW(x,y) \
258:  if (new_count++ < wscount) \
259:  { \
260:    next_new_state->offset = (x); \
261:    next_new_state->count = (y); \
262:    next_new_state->ims = ims; \
263:    next_new_state++; \
264:    DPRINTF(("%.sADD_NEW(%d,%d) \n", rlevel*2-2, SP, (x), (y))); \
265:  } \
266:  else return PCRE_ERROR_DFA_WSSIZE
267:
268: #define ADD_NEW_DATA(x,y,z) \
269:  if (new_count++ < wscount) \
270:  { \
271:    next_new_state->offset = (x); \
272:    next_new_state->count = (y); \
273:    next_new_state->ims = ims; \
274:    next_new_state->data = (z); \
275:    next_new_state++; \
276:    DPRINTF(("%.sADD_NEW_DATA(%d,%d,%d) \n", rlevel*2-2, SP, (x), (y), (z))); \
277:  } \
278:  else return PCRE_ERROR_DFA_WSSIZE
279:
280: /* And now, here is the code */
281:

```

```

282: static int
283: internal_dfa_exec(
284:   dfa_match_data *md,
285:   const uschar *this_start_code,
286:   const uschar *current_subject,
287:   int start_offset,
288:   int *offsets,
289:   int offsetcount,
290:   int *workspace,
291:   int wscount,
292:   int ims,
293:   int rlevel,
294:   int recursing)
295: {
296:   stateblock *active_states, *new_states, *temp_states;
297:   stateblock *next_active_state, *next_new_state;
298:
299:   const uschar *ctypes, *lcc, *fcc;
300:   const uschar *ptr;
301:   const uschar *end_code, *first_op;
302:
303:   int active_count, new_count, match_count;
304:
305:   /* Some fields in the md block are frequently referenced, so we load them into
306:   independent variables in the hope that this will perform better. */
307:
308:   const uschar *start_subject = md->start_subject;
309:   const uschar *end_subject = md->end_subject;
310:   const uschar *start_code = md->start_code;
311:
312:   #ifdef SUPPORT_UTF8
313:   BOOL utf8 = (md->poptions & PCRE_UTF8) != 0;
314:   #else
315:   BOOL utf8 = FALSE;
316:   #endif
317:
318:   rlevel++;
319:   offsetcount &= (-2);
320:
321:   wscount -= 2;
322:   wscount = (wscount - (wscount % (INTS_PER_STATEBLOCK * 2))) /

```

```

323:     (2 * INTS_PER_STATEBLOCK);
324:
325: DPRINTF((" \n%.*s----- \n"
326: "%.sCall to internal_dfa_exec f=%d r=%d \n",
327: rlevel*2-2, SP, rlevel*2-2, SP, rlevel, recursing));
328:
329: ctypes = md->tables + ctypes_offset;
330: lcc = md->tables + lcc_offset;
331: fcc = md->tables + fcc_offset;
332:
333: match_count = PCRE_ERROR_NOMATCH; /* A negative number */
334:
335: active_states = (stateblock*)(workspace + 2);
336: next_new_state = new_states = active_states + wscount;
337: new_count = 0;
338:
339: first_op = this_start_code + 1 + LINK_SIZE +
340: ((*this_start_code == OP_CBRA || *this_start_code == OP_SCBRA)? 2:0);
341:
342: /* The first thing in any (sub) pattern is a bracket of some sort. Push all
343: the alternative states onto the list, and find out where the end is. This
344: makes is possible to use this function recursively, when we want to stop at a
345: matching internal ket rather than at the end.
346:
347: If the first opcode in the first alternative is OP_REVERSE, we are dealing with
348: a backward assertion. In that case, we have to find out the maximum amount to
349: move back, and set up each alternative appropriately. */
350:
351: if (*first_op == OP_REVERSE)
352: {
353:   int max_back = 0;
354:   int gone_back;
355:
356:   end_code = this_start_code;
357:   do
358:   {
359:     int back = GET(end_code, 2+LINK_SIZE);
360:     if (back > max_back) max_back = back;
361:     end_code += GET(end_code, 1);
362:   }
363:   while (*end_code == OP_ALT);

```

```

364:
365: /* If we can't go back the amount required for the longest lookbehind
366: pattern, go back as far as we can; some alternatives may still be viable. */
367:
368: #ifdef SUPPORT_UTF8
369: /* In character mode we have to step back character by character */
370:
371: if (utf8)
372: {
373:     for (gone_back = 0; gone_back < max_back; gone_back++)
374:     {
375:         if (current_subject <= start_subject) break;
376:         current_subject--;
377:         while (current_subject > start_subject &&
378:             (*current_subject & 0xc0) == 0x80)
379:             current_subject--;
380:     }
381: }
382: else
383: #endif
384:
385: /* In byte-mode we can do this quickly. */
386:
387: {
388:     gone_back = (current_subject - max_back < start_subject)?
389:         current_subject - start_subject : max_back;
390:     current_subject -= gone_back;
391: }
392:
393: /* Now we can process the individual branches. */
394:
395: end_code = this_start_code;
396: do
397: {
398:     int back = GET(end_code, 2+LINK_SIZE);
399:     if (back <= gone_back)
400:     {
401:         int bstate = end_code - start_code + 2 + 2*LINK_SIZE;
402:         ADD_NEW_DATA(-bstate, 0, gone_back - back);
403:     }
404:     end_code += GET(end_code, 1);

```

```

405:  }
406:  while (*end_code == OP_ALT);
407:  }
408:
409: /* This is the code for a "normal" subpattern (not a backward assertion). The
410: start of a whole pattern is always one of these. If we are at the top level,
411: we may be asked to restart matching from the same point that we reached for a
412: previous partial match. We still have to scan through the top-level branches to
413: find the end state. */
414:
415: else
416: {
417:   end_code = this_start_code;
418:
419:   /* Restarting */
420:
421:   if (rlevel == 1 && (md->moptions & PCRE_DFA_RESTART) != 0)
422:   {
423:     do { end_code += GET(end_code, 1); } while (*end_code == OP_ALT);
424:     new_count = workspace[1];
425:     if (!workspace[0])
426:       memcpy(new_states, active_states, new_count * sizeof(stateblock));
427:   }
428:
429:   /* Not restarting */
430:
431:   else
432:   {
433:     int length = 1 + LINK_SIZE +
434:       ((*this_start_code == OP_CBRA || *this_start_code == OP_SCBRA)? 2:0);
435:     do
436:     {
437:       ADD_NEW(end_code - start_code + length, 0);
438:       end_code += GET(end_code, 1);
439:       length = 1 + LINK_SIZE;
440:     }
441:     while (*end_code == OP_ALT);
442:   }
443: }
444:
445: workspace[0] = 0; /* Bit indicating which vector is current */

```



```

446:
447: DPRINTF(("%. *sEnd state = %d \n", rlevel*2-2, SP, end_code - start_code));
448:
449: /* Loop for scanning the subject */
450:
451: ptr = current_subject;
452: for (;;)
453: {
454:   int i, j;
455:   int clen, dlen;
456:   unsigned int c, d;
457:
458:   /* Make the new state list into the active state list and empty the
459:   new state list. */
460:
461:   temp_states = active_states;
462:   active_states = new_states;
463:   new_states = temp_states;
464:   active_count = new_count;
465:   new_count = 0;
466:
467:   workspace[0] ^= 1;          /* Remember for the restarting feature */
468:   workspace[1] = active_count;
469:
470: #ifdef DEBUG
471:   printf("%. *sNext character: rest of subject = \'", rlevel*2-2, SP);
472:   pchars((uschar *)ptr, strlen((char *)ptr), stdout);
473:   printf("\' \n");
474:
475:   printf("%. *sActive states: ", rlevel*2-2, SP);
476:   for (i = 0; i < active_count; i++)
477:     printf("%d/%d ", active_states[i].offset, active_states[i].count);
478:   printf(" \n");
479: #endif
480:
481:   /* Set the pointers for adding new states */
482:
483:   next_active_state = active_states + active_count;
484:   next_new_state = new_states;
485:
486:   /* Load the current character from the subject outside the loop, as many

```

```

487: different states may want to look at it, and we assume that at least one
488: will. */
489:
490: if (ptr < end_subject)
491: {
492:     clen = 1;    /* Number of bytes in the character */
493: #ifdef SUPPORT_UTF8
494:     if (utf8) { GETCHARLEN(c, ptr, clen); } else
495: #endif /* SUPPORT_UTF8 */
496:     c = *ptr;
497: }
498: else
499: {
500:     clen = 0;    /* This indicates the end of the subject */
501:     c = NOTACHAR; /* This value should never actually be used */
502: }
503:
504: /* Scan up the active states and act on each one. The result of an action
505: may be to add more states to the currently active list (e.g. on hitting a
506: parenthesis) or it may be to put states on the new list, for considering
507: when we move the character pointer on. */
508:
509: for (i = 0; i < active_count; i++)
510: {
511:     stateblock *current_state = active_states + i;
512:     const uschar *code;
513:     int state_offset = current_state->offset;
514:     int count, codevalue;
515:
516: #ifdef DEBUG
517:     printf ("%sProcessing state %d c=", rlevel*2-2, SP, state_offset);
518:     if (clen == 0) printf("EOL \n");
519:     else if (c > 32 && c < 127) printf("'%c' \n", c);
520:     else printf("0x%02x \n", c);
521: #endif
522:
523:     /* This variable is referred to implicitly in the ADD_xxx macros. */
524:
525:     ims = current_state->ims;
526:
527:     /* A negative offset is a special case meaning "hold off going to this

```

```

528: (negated) state until the number of characters in the data field have
529: been skipped". */
530:
531: if (state_offset < 0)
532: {
533:     if (current_state->data > 0)
534:     {
535:         DPRINTF(("%.sSkipping this character \n", rlevel*2-2, SP));
536:         ADD_NEW_DATA(state_offset, current_state->count,
537:             current_state->data - 1);
538:         continue;
539:     }
540:     else
541:     {
542:         current_state->offset = state_offset = -state_offset;
543:     }
544: }
545:
546: /* Check for a duplicate state with the same count, and skip if found. */
547:
548: for (j = 0; j < i; j++)
549: {
550:     if (active_states[j].offset == state_offset &&
551:         active_states[j].count == current_state->count)
552:     {
553:         DPRINTF(("%.sDuplicate state: skipped \n", rlevel*2-2, SP));
554:         goto NEXT_ACTIVE_STATE;
555:     }
556: }
557:
558: /* The state offset is the offset to the opcode */
559:
560: code = start_code + state_offset;
561: codevalue = *code;
562:
563: /* If this opcode is followed by an inline character, load it. It is
564: tempting to test for the presence of a subject character here, but that
565: is wrong, because sometimes zero repetitions of the subject are
566: permitted.
567:
568: We also use this mechanism for opcodes such as OP_TYPEPLUS that take an

```

```

569: argument that is not a data character - but is always one byte long. We
570: have to take special action to deal with \P, \p, \H, \h, \V, \v and \X in
571: this case. To keep the other cases fast, convert these ones to new opcodes.
572: */
573:
574: if (coptable[codevalue] > 0)
575: {
576:     dlen = 1;
577: #ifdef SUPPORT_UTF8
578:     if (utf8) { GETCHARLEN(d, (code + coptable[codevalue]), dlen); } else
579: #endif /* SUPPORT_UTF8 */
580:     d = code[coptable[codevalue]];
581:     if (codevalue >= OP_TYPESTAR)
582:     {
583:         switch(d)
584:         {
585:             case OP_ANYBYTE: return PCRE_ERROR_DFA_UITEM;
586:             case OP_NOTPROP:
587:             case OP_PROP: codevalue += OP_PROP_EXTRA; break;
588:             case OP_ANYNL: codevalue += OP_ANYNL_EXTRA; break;
589:             case OP_EXTUNI: codevalue += OP_EXTUNI_EXTRA; break;
590:             case OP_NOT_HSPACE:
591:             case OP_HSPACE: codevalue += OP_HSPACE_EXTRA; break;
592:             case OP_NOT_VSPACE:
593:             case OP_VSPACE: codevalue += OP_VSPACE_EXTRA; break;
594:             default: break;
595:         }
596:     }
597: }
598: else
599: {
600:     dlen = 0;      /* Not strictly necessary, but compilers moan */
601:     d = NOTACHAR; /* if these variables are not set. */
602: }
603:
604:
605: /* Now process the individual opcodes */
606:
607: switch (codevalue)
608: {
609:

```

```

610:                                                                 /*
=====
*/
611:  /* Reached a closing bracket. If not at the end of the pattern, carry
612:  on with the next opcode. Otherwise, unless we have an empty string and
613:  PCRE_NOTEMPTY is set, save the match data, shifting up all previous
614:  matches so we always have the longest first. */
615:
616:  case OP_KET:
617:  case OP_KETRMIN:
618:  case OP_KETRMX:
619:  if (code != end_code)
620:  {
621:    ADD_ACTIVE(state_offset + 1 + LINK_SIZE, 0);
622:    if (codevalue != OP_KET)
623:    {
624:      ADD_ACTIVE(state_offset - GET(code, 1), 0);
625:    }
626:  }
627:  else if (ptr > current_subject || (md->moptions & PCRE_NOTEMPTY) == 0)
628:  {
629:    if (match_count < 0) match_count = (offsetcount >= 2)? 1 : 0;
630:    else if (match_count > 0 && ++match_count * 2 >= offsetcount)
631:      match_count = 0;
632:    count = ((match_count == 0)? offsetcount : match_count * 2) - 2;
633:    if (count > 0) memmove(offsets + 2, offsets, count * sizeof(int));
634:    if (offsetcount >= 2)
635:    {
636:      offsets[0] = current_subject - start_subject;
637:      offsets[1] = ptr - start_subject;
638:      DPRINTF(("%.sSet matched string = \"%.s \"\n", rlevel*2-2, SP,
639:        offsets[1] - offsets[0], current_subject));
640:    }
641:    if ((md->moptions & PCRE_DFA_SHORTEST) != 0)
642:    {
643:      DPRINTF(("%.sEnd of internal_dfa_exec %d: returning %d\n"
644:        "%.s----- \n\n", rlevel*2-2, SP, rlevel,
645:        match_count, rlevel*2-2, SP));
646:      return match_count;
647:    }
648:  }

```

```

649:    break;
650:
651:
=====
*/
652:    /* These opcodes add to the current list of states without looking
653:    at the current character. */
654:
655:    /*-----*/
656:    case OP_ALT:
657:    do { code += GET(code, 1); } while (*code == OP_ALT);
658:    ADD_ACTIVE(code - start_code, 0);
659:    break;
660:
661:    /*-----*/
662:    case OP_BRA:
663:    case OP_SBRA:
664:    do
665:    {
666:        ADD_ACTIVE(code - start_code + 1 + LINK_SIZE, 0);
667:        code += GET(code, 1);
668:    }
669:    while (*code == OP_ALT);
670:    break;
671:
672:    /*-----*/
673:    case OP_CBRA:
674:    case OP_SCBRA:
675:    ADD_ACTIVE(code - start_code + 3 + LINK_SIZE, 0);
676:    code += GET(code, 1);
677:    while (*code == OP_ALT)
678:    {
679:        ADD_ACTIVE(code - start_code + 1 + LINK_SIZE, 0);
680:        code += GET(code, 1);
681:    }
682:    break;
683:
684:    /*-----*/
685:    case OP_BRAZERO:
686:    case OP_BRAMINZERO:
687:    ADD_ACTIVE(state_offset + 1, 0);

```

```

688: code += 1 + GET(code, 2);
689: while (*code == OP_ALT) code += GET(code, 1);
690: ADD_ACTIVE(code - start_code + 1 + LINK_SIZE, 0);
691: break;
692:
693: /*-----*/
694: case OP_SKIPZERO:
695: code += 1 + GET(code, 2);
696: while (*code == OP_ALT) code += GET(code, 1);
697: ADD_ACTIVE(code - start_code + 1 + LINK_SIZE, 0);
698: break;
699:
700: /*-----*/
701: case OP_CIRC:
702: if ((ptr == start_subject && (md->moptions & PCRE_NOTBOL) == 0) ||
703:     ((ims & PCRE_MULTILINE) != 0 &&
704:      ptr != end_subject &&
705:      WAS_NEWLINE(ptr)))
706:     { ADD_ACTIVE(state_offset + 1, 0); }
707: break;
708:
709: /*-----*/
710: case OP_EOD:
711: if (ptr >= end_subject) { ADD_ACTIVE(state_offset + 1, 0); }
712: break;
713:
714: /*-----*/
715: case OP_OPT:
716: ims = code[1];
717: ADD_ACTIVE(state_offset + 2, 0);
718: break;
719:
720: /*-----*/
721: case OP_SOD:
722: if (ptr == start_subject) { ADD_ACTIVE(state_offset + 1, 0); }
723: break;
724:
725: /*-----*/
726: case OP_SOM:
727: if (ptr == start_subject + start_offset) { ADD_ACTIVE(state_offset + 1, 0); }
728: break;

```

```

729:
730:
731:                                                     /*
=====
*/
732:  /* These opcodes inspect the next subject character, and sometimes
733:  the previous one as well, but do not have an argument. The variable
734:  clen contains the length of the current character and is zero if we are
735:  at the end of the subject. */
736:
737:  /*-----*/
738:  case OP_ANY:
739:  if (clen > 0 && !IS_NEWLINE(ptr))
740:    { ADD_NEW(state_offset + 1, 0); }
741:  break;
742:
743:  /*-----*/
744:  case OP_ALLANY:
745:  if (clen > 0)
746:    { ADD_NEW(state_offset + 1, 0); }
747:  break;
748:
749:  /*-----*/
750:  case OP_EODN:
751:  if (clen == 0 || (IS_NEWLINE(ptr) && ptr == end_subject - md->nllen))
752:    { ADD_ACTIVE(state_offset + 1, 0); }
753:  break;
754:
755:  /*-----*/
756:  case OP_DOLL:
757:  if ((md->moptions & PCRE_NOTEOL) == 0)
758:    {
759:    if (clen == 0 ||
760:        (IS_NEWLINE(ptr) &&
761:         ((ims & PCRE_MULTILINE) != 0 || ptr == end_subject - md->nllen)
762:        ))
763:      { ADD_ACTIVE(state_offset + 1, 0); }
764:    }
765:  else if ((ims & PCRE_MULTILINE) != 0 && IS_NEWLINE(ptr))
766:    { ADD_ACTIVE(state_offset + 1, 0); }
767:  break;

```



```

768:
769:  /*-----*/
770:
771:  case OP_DIGIT:
772:  case OP_WHITESPACE:
773:  case OP_WORDCHAR:
774:  if (clen > 0 && c < 256 &&
775:      ((ctypes[c] & toptable1[codevalue]) ^ toptable2[codevalue]) != 0)
776:      { ADD_NEW(state_offset + 1, 0); }
777:  break;
778:
779:  /*-----*/
780:  case OP_NOT_DIGIT:
781:  case OP_NOT_WHITESPACE:
782:  case OP_NOT_WORDCHAR:
783:  if (clen > 0 && (c >= 256 ||
784:      ((ctypes[c] & toptable1[codevalue]) ^ toptable2[codevalue]) != 0))
785:      { ADD_NEW(state_offset + 1, 0); }
786:  break;
787:
788:  /*-----*/
789:  case OP_WORD_BOUNDARY:
790:  case OP_NOT_WORD_BOUNDARY:
791:  {
792:      int left_word, right_word;
793:
794:      if (ptr > start_subject)
795:      {
796:          const uschar *temp = ptr - 1;
797: #ifdef SUPPORT_UTF8
798:          if (utf8) BACKCHAR(temp);
799: #endif
800:          GETCHARTEST(d, temp);
801:          left_word = d < 256 && (ctypes[d] & ctype_word) != 0;
802:      }
803:      else left_word = 0;
804:
805:      if (clen > 0) right_word = c < 256 && (ctypes[c] & ctype_word) != 0;
806:      else right_word = 0;
807:
808:      if ((left_word == right_word) == (codevalue == OP_NOT_WORD_BOUNDARY))

```

```

809:     { ADD_ACTIVE(state_offset + 1, 0); }
810: }
811: break;
812:
813:
814: /*-----*/
815: /* Check the next character by Unicode property. We will get here only
816:    if the support is in the binary; otherwise a compile-time error occurs.
817:    */
818:
819: #ifdef SUPPORT_UCP
820:     case OP_PROP:
821:     case OP_NOTPROP:
822:     if (clen > 0)
823:     {
824:         BOOL OK;
825:         const ucd_record * prop = GET_UCD(c);
826:         switch(code[1])
827:         {
828:             case PT_ANY:
829:                 OK = TRUE;
830:                 break;
831:
832:             case PT_LAMP:
833:                 OK = prop->chartype == ucp_Lu || prop->chartype == ucp_Ll || prop->chartype == ucp_Lt;
834:                 break;
835:
836:             case PT_GC:
837:                 OK = _pcre_ucp_gentype[prop->chartype] == code[2];
838:                 break;
839:
840:             case PT_PC:
841:                 OK = prop->chartype == code[2];
842:                 break;
843:
844:             case PT_SC:
845:                 OK = prop->script == code[2];
846:                 break;
847:
848:             /* Should never occur, but keep compilers from grumbling. */
849:

```

```

850:     default:
851:     OK = codevalue != OP_PROP;
852:     break;
853:     }
854:
855:     if (OK == (codevalue == OP_PROP)) { ADD_NEW(state_offset + 3, 0); }
856:     }
857:     break;
858: #endif
859:
860:
861:
862:
863:     /* These opcodes likewise inspect the subject character, but have an
864:     argument that is not a data character. It is one of these opcodes:
865:         OP_ANY, OP_ALLANY, OP_DIGIT, OP_NOT_DIGIT, OP_WHITESPACE,
866:         OP_NOT_SPACE,
867:         OP_WORDCHAR, OP_NOT_WORDCHAR. The value is loaded into d. */
868:     case OP_TYPEPLUS:
869:     case OP_TYPEMINPLUS:
870:     case OP_TYPEPOSPLUS:
871:     count = current_state->count; /* Already matched */
872:     if (count > 0) { ADD_ACTIVE(state_offset + 2, 0); }
873:     if (c < 0)
874:     {
875:     if ((c >= 256 && d != OP_DIGIT && d != OP_WHITESPACE && d != OP_WORDCHAR) ||
876:         (c < 256 &&
877:          (d != OP_ANY || !IS_NEWLINE(ptr)) &&
878:          ((ctypes[c] & toptable1[d]) ^ toptable2[d]) != 0))
879:     {
880:     if (count > 0 && codevalue == OP_TYPEPOSPLUS)
881:     {
882:     active_count--; /* Remove non-match possibility */
883:     next_active_state--;
884:     }
885:     count++;
886:     ADD_NEW(state_offset, count);
887:     }
888:     }

```

```

889: break;
890:
891: /*-----*/
892: case OP_TYPEQUERY:
893: case OP_TYPEMINQUERY:
894: case OP_TYPEPOSQUERY:
895: ADD_ACTIVE(state_offset + 2, 0);
896: if (clen > 0)
897: {
898:   if ((c >= 256 && d != OP_DIGIT && d != OP_WHITESPACE && d != OP_WORDCHAR) ||
899:       (c < 256 &&
900:        (d != OP_ANY || !IS_NEWLINE(ptr)) &&
901:        ((ctypes[c] & toptable1[d]) ^ toptable2[d]) != 0))
902:   {
903:     if (codevalue == OP_TYPEPOSQUERY)
904:     {
905:       active_count--;      /* Remove non-match possibility */
906:       next_active_state--;
907:     }
908:     ADD_NEW(state_offset + 2, 0);
909:   }
910: }
911: break;
912:
913: /*-----*/
914: case OP_TYPESTAR:
915: case OP_TYPEMINSTAR:
916: case OP_TYPEPOSSTAR:
917: ADD_ACTIVE(state_offset + 2, 0);
918: if (clen > 0)
919: {
920:   if ((c >= 256 && d != OP_DIGIT && d != OP_WHITESPACE && d != OP_WORDCHAR) ||
921:       (c < 256 &&
922:        (d != OP_ANY || !IS_NEWLINE(ptr)) &&
923:        ((ctypes[c] & toptable1[d]) ^ toptable2[d]) != 0))
924:   {
925:     if (codevalue == OP_TYPEPOSSTAR)
926:     {
927:       active_count--;      /* Remove non-match possibility */
928:       next_active_state--;
929:     }

```

```

930:     ADD_NEW(state_offset, 0);
931: }
932: }
933: break;
934:
935: /*-----*/
936: case OP_TYPEEXACT:
937:     count = current_state->count; /* Number already matched */
938:     if (clen > 0)
939:     {
940:         if ((c >= 256 && d != OP_DIGIT && d != OP_WHITESPACE && d != OP_WORDCHAR) ||
941:             (c < 256 &&
942:              (d != OP_ANY || !IS_NEWLINE(ptr)) &&
943:              ((ctypes[c] & toptable1[d]) ^ toptable2[d]) != 0))
944:         {
945:             if (++count >= GET2(code, 1))
946:                 { ADD_NEW(state_offset + 4, 0); }
947:             else
948:                 { ADD_NEW(state_offset, count); }
949:         }
950:     }
951:     break;
952:
953: /*-----*/
954: case OP_TYPEUPTO:
955: case OP_TYPEMINUPTO:
956: case OP_TYPEPOSUPTO:
957:     ADD_ACTIVE(state_offset + 4, 0);
958:     count = current_state->count; /* Number already matched */
959:     if (clen > 0)
960:     {
961:         if ((c >= 256 && d != OP_DIGIT && d != OP_WHITESPACE && d != OP_WORDCHAR) ||
962:             (c < 256 &&
963:              (d != OP_ANY || !IS_NEWLINE(ptr)) &&
964:              ((ctypes[c] & toptable1[d]) ^ toptable2[d]) != 0))
965:         {
966:             if (codevalue == OP_TYPEPOSUPTO)
967:             {
968:                 active_count--; /* Remove non-match possibility */
969:                 next_active_state--;
970:             }

```

```

971:     if (++count >= GET2(code, 1))
972:         { ADD_NEW(state_offset + 4, 0); }
973:     else
974:         { ADD_NEW(state_offset, count); }
975:     }
976: }
977: break;
978:
979:
=====
*/
980: /* These are virtual opcodes that are used when something like
981: OP_TYPEPLUS has OP_PROP, OP_NOTPROP, OP_ANYNL, or OP_EXTUNI as its
982: argument. It keeps the code above fast for the other cases. The argument
983: is in the d variable. */
984:
985: #ifdef SUPPORT_UCP
986:     case OP_PROP_EXTRA + OP_TYPEPLUS:
987:     case OP_PROP_EXTRA + OP_TYPEMINPLUS:
988:     case OP_PROP_EXTRA + OP_TYPEPOSPLUS:
989:         count = current_state->count; /* Already matched */
990:         if (count > 0) { ADD_ACTIVE(state_offset + 4, 0); }
991:         if (clen > 0)
992:         {
993:             BOOL OK;
994:             const ucd_record * prop = GET_UCD(c);
995:             switch(code[2])
996:             {
997:                 case PT_ANY:
998:                     OK = TRUE;
999:                     break;
1000:
1001:                 case PT_LAMP:
1002:                     OK = prop->chartype == ucp_Lu || prop->chartype == ucp_Ll || prop->chartype == ucp_Lt;
1003:                     break;
1004:
1005:                 case PT_GC:
1006:                     OK = _pcre_ucp_gentype[prop->chartype] == code[3];
1007:                     break;
1008:
1009:                 case PT_PC:

```

```

1010:     OK = prop->chartype == code[3];
1011:     break;
1012:
1013:     case PT_SC:
1014:     OK = prop->script == code[3];
1015:     break;
1016:
1017:     /* Should never occur, but keep compilers from grumbling. */
1018:
1019:     default:
1020:     OK = codevalue != OP_PROP;
1021:     break;
1022:     }
1023:
1024:     if (OK == (d == OP_PROP))
1025:     {
1026:     if (count > 0 && codevalue == OP_PROP_EXTRA + OP_TYPEPOSPLUS)
1027:     {
1028:     active_count--;      /* Remove non-match possibility */
1029:     next_active_state--;
1030:     }
1031:     count++;
1032:     ADD_NEW(state_offset, count);
1033:     }
1034:     }
1035:     break;
1036:
1037:     /*-----*/
1038:     case OP_EXTUNI_EXTRA + OP_TYPEPLUS:
1039:     case OP_EXTUNI_EXTRA + OP_TYPEMINPLUS:
1040:     case OP_EXTUNI_EXTRA + OP_TYPEPOSPLUS:
1041:     count = current_state->count; /* Already matched */
1042:     if (count > 0) { ADD_ACTIVE(state_offset + 2, 0); }
1043:     if (clen > 0 && UCD_CATEGORY(c) != ucp_M)
1044:     {
1045:     const uschar *nptr = ptr + clen;
1046:     int ncount = 0;
1047:     if (count > 0 && codevalue == OP_EXTUNI_EXTRA + OP_TYPEPOSPLUS)
1048:     {
1049:     active_count--;      /* Remove non-match possibility */
1050:     next_active_state--;

```

```

1051:     }
1052:     while (nptr < end_subject)
1053:     {
1054:         int nd;
1055:         int ndlen = 1;
1056:         GETCHARLEN(nd, nptr, ndlen);
1057:         if (UCD_CATEGORY(nd) != ucp_M) break;
1058:         ncount++;
1059:         nptr += ndlen;
1060:     }
1061:     count++;
1062:     ADD_NEW_DATA(-state_offset, count, ncount);
1063: }
1064: break;
1065: #endif
1066:
1067: /*-----*/
1068: case OP_ANYNL_EXTRA + OP_TYPEPLUS:
1069: case OP_ANYNL_EXTRA + OP_TYPEMINPLUS:
1070: case OP_ANYNL_EXTRA + OP_TYPEPOSPLUS:
1071:     count = current_state->count; /* Already matched */
1072:     if (count > 0) { ADD_ACTIVE(state_offset + 2, 0); }
1073:     if (clen > 0)
1074:     {
1075:         int ncount = 0;
1076:         switch (c)
1077:         {
1078:             case 0x000b:
1079:             case 0x000c:
1080:             case 0x0085:
1081:             case 0x2028:
1082:             case 0x2029:
1083:                 if ((md->moptions & PCRE_BSR_ANYCRLF) != 0) break;
1084:                 goto ANYNL01;
1085:
1086:             case 0x000d:
1087:                 if (ptr + 1 < end_subject && ptr[1] == 0x0a) ncount = 1;
1088:                 /* Fall through */
1089:
1090:             ANYNL01:
1091:             case 0x000a:

```



```

1092:     if (count > 0 && codevalue == OP_ANYNL_EXTRA + OP_TYPEPOSPLUS)
1093:     {
1094:         active_count--;      /* Remove non-match possibility */
1095:         next_active_state--;
1096:     }
1097:     count++;
1098:     ADD_NEW_DATA(-state_offset, count, ncount);
1099:     break;
1100:
1101:     default:
1102:     break;
1103: }
1104: }
1105: break;
1106:
1107: /*-----*/
1108: case OP_VSPACE_EXTRA + OP_TYPEPLUS:
1109: case OP_VSPACE_EXTRA + OP_TYPEMINPLUS:
1110: case OP_VSPACE_EXTRA + OP_TYPEPOSPLUS:
1111:     count = current_state->count; /* Already matched */
1112:     if (count > 0) { ADD_ACTIVE(state_offset + 2, 0); }
1113:     if (klen > 0)
1114:     {
1115:         BOOL OK;
1116:         switch (c)
1117:         {
1118:             case 0x000a:
1119:             case 0x000b:
1120:             case 0x000c:
1121:             case 0x000d:
1122:             case 0x0085:
1123:             case 0x2028:
1124:             case 0x2029:
1125:                 OK = TRUE;
1126:                 break;
1127:
1128:             default:
1129:                 OK = FALSE;
1130:                 break;
1131:         }
1132:

```

```

1133:     if (OK == (d == OP_VSPACE))
1134:     {
1135:         if (count > 0 && codevalue == OP_VSPACE_EXTRA + OP_TYPEPOSPLUS)
1136:         {
1137:             active_count--;      /* Remove non-match possibility */
1138:             next_active_state--;
1139:         }
1140:         count++;
1141:         ADD_NEW_DATA(-state_offset, count, 0);
1142:     }
1143: }
1144: break;
1145:
1146: /*-----*/
1147: case OP_HSPACE_EXTRA + OP_TYPEPLUS:
1148: case OP_HSPACE_EXTRA + OP_TYPEMINPLUS:
1149: case OP_HSPACE_EXTRA + OP_TYPEPOSPLUS:
1150:     count = current_state->count; /* Already matched */
1151:     if (count > 0) { ADD_ACTIVE(state_offset + 2, 0); }
1152:     if (cflen > 0)
1153:     {
1154:         BOOL OK;
1155:         switch (c)
1156:         {
1157:             case 0x09:    /* HT */
1158:             case 0x20:    /* SPACE */
1159:             case 0xa0:    /* NBSP */
1160:             case 0x1680:  /* OGHAM SPACE MARK */
1161:             case 0x180e:  /* MONGOLIAN VOWEL SEPARATOR */
1162:             case 0x2000:  /* EN QUAD */
1163:             case 0x2001:  /* EM QUAD */
1164:             case 0x2002:  /* EN SPACE */
1165:             case 0x2003:  /* EM SPACE */
1166:             case 0x2004:  /* THREE-PER-EM SPACE */
1167:             case 0x2005:  /* FOUR-PER-EM SPACE */
1168:             case 0x2006:  /* SIX-PER-EM SPACE */
1169:             case 0x2007:  /* FIGURE SPACE */
1170:             case 0x2008:  /* PUNCTUATION SPACE */
1171:             case 0x2009:  /* THIN SPACE */
1172:             case 0x200A:  /* HAIR SPACE */
1173:             case 0x202f:  /* NARROW NO-BREAK SPACE */

```

```

1174:     case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
1175:     case 0x3000: /* IDEOGRAPHIC SPACE */
1176:         OK = TRUE;
1177:         break;
1178:
1179:     default:
1180:         OK = FALSE;
1181:         break;
1182:     }
1183:
1184:     if (OK == (d == OP_HSPACE))
1185:     {
1186:         if (count > 0 && codevalue == OP_HSPACE_EXTRA + OP_TYPEPOSPLUS)
1187:         {
1188:             active_count--; /* Remove non-match possibility */
1189:             next_active_state--;
1190:         }
1191:         count++;
1192:         ADD_NEW_DATA(-state_offset, count, 0);
1193:     }
1194: }
1195: break;
1196:
1197: /*-----*/
1198: #ifdef SUPPORT_UCP
1199:     case OP_PROP_EXTRA + OP_TYPEQUERY:
1200:     case OP_PROP_EXTRA + OP_TYPEMINQUERY:
1201:     case OP_PROP_EXTRA + OP_TYPEPOSQUERY:
1202:         count = 4;
1203:         goto QS1;
1204:
1205:     case OP_PROP_EXTRA + OP_TYPESTAR:
1206:     case OP_PROP_EXTRA + OP_TYPEMINSTAR:
1207:     case OP_PROP_EXTRA + OP_TYPEPOSSTAR:
1208:         count = 0;
1209:
1210:     QS1:
1211:
1212:         ADD_ACTIVE(state_offset + 4, 0);
1213:         if (clen > 0)
1214:         {

```

```

1215:     BOOL OK;
1216:     const ucd_record * prop = GET_UCD(c);
1217:     switch(code[2])
1218:     {
1219:     case PT_ANY:
1220:         OK = TRUE;
1221:         break;
1222:
1223:     case PT_LAMP:
1224:         OK = prop->chartype == ucp_Lu || prop->chartype == ucp_Ll || prop->chartype == ucp_Lt;
1225:         break;
1226:
1227:     case PT_GC:
1228:         OK = _pcre_ucp_gentype[prop->chartype] == code[3];
1229:         break;
1230:
1231:     case PT_PC:
1232:         OK = prop->chartype == code[3];
1233:         break;
1234:
1235:     case PT_SC:
1236:         OK = prop->script == code[3];
1237:         break;
1238:
1239:         /* Should never occur, but keep compilers from grumbling. */
1240:
1241:     default:
1242:         OK = codevalue != OP_PROP;
1243:         break;
1244:     }
1245:
1246:     if (OK == (d == OP_PROP))
1247:     {
1248:         if (codevalue == OP_PROP_EXTRA + OP_TYPEPOSSTAR ||
1249:             codevalue == OP_PROP_EXTRA + OP_TYPEPOSQUERY)
1250:         {
1251:             active_count--;          /* Remove non-match possibility */
1252:             next_active_state--;
1253:         }
1254:         ADD_NEW(state_offset + count, 0);
1255:     }

```

```

1256:     }
1257:     break;
1258:
1259:     /*-----*/
1260:     case OP_EXTUNI_EXTRA + OP_TYPEQUERY:
1261:     case OP_EXTUNI_EXTRA + OP_TYPEMINQUERY:
1262:     case OP_EXTUNI_EXTRA + OP_TYPEPOSQUERY:
1263:         count = 2;
1264:         goto QS2;
1265:
1266:     case OP_EXTUNI_EXTRA + OP_TYPESTAR:
1267:     case OP_EXTUNI_EXTRA + OP_TYPEMINSTAR:
1268:     case OP_EXTUNI_EXTRA + OP_TYPEPOSSTAR:
1269:         count = 0;
1270:
1271:     QS2:
1272:
1273:     ADD_ACTIVE(state_offset + 2, 0);
1274:     if (clen > 0 && UCD_CATEGORY(c) != ucp_M)
1275:     {
1276:         const uschar *nptr = ptr + clen;
1277:         int ncount = 0;
1278:         if (codevalue == OP_EXTUNI_EXTRA + OP_TYPEPOSSTAR ||
1279:             codevalue == OP_EXTUNI_EXTRA + OP_TYPEPOSQUERY)
1280:         {
1281:             active_count--;      /* Remove non-match possibility */
1282:             next_active_state--;
1283:         }
1284:         while (nptr < end_subject)
1285:         {
1286:             int nd;
1287:             int ndlen = 1;
1288:             GETCHARLEN(nd, nptr, ndlen);
1289:             if (UCD_CATEGORY(nd) != ucp_M) break;
1290:             ncount++;
1291:             nptr += ndlen;
1292:         }
1293:         ADD_NEW_DATA(-(state_offset + count), 0, ncount);
1294:     }
1295:     break;
1296: #endif

```

```

1297:
1298:  /*-----*/
1299:  case OP_ANYNL_EXTRA + OP_TYPEQUERY:
1300:  case OP_ANYNL_EXTRA + OP_TYPEMINQUERY:
1301:  case OP_ANYNL_EXTRA + OP_TYPEPOSQUERY:
1302:    count = 2;
1303:    goto QS3;
1304:
1305:  case OP_ANYNL_EXTRA + OP_TYPESTAR:
1306:  case OP_ANYNL_EXTRA + OP_TYPEMINSTAR:
1307:  case OP_ANYNL_EXTRA + OP_TYPEPOSSTAR:
1308:    count = 0;
1309:
1310:  QS3:
1311:    ADD_ACTIVE(state_offset + 2, 0);
1312:    if (clen > 0)
1313:    {
1314:      int ncount = 0;
1315:      switch (c)
1316:      {
1317:        case 0x000b:
1318:        case 0x000c:
1319:        case 0x0085:
1320:        case 0x2028:
1321:        case 0x2029:
1322:          if ((md->moptions & PCRE_BSR_ANYCRLF) != 0) break;
1323:          goto ANYNL02;
1324:
1325:        case 0x000d:
1326:          if (ptr + 1 < end_subject && ptr[1] == 0x0a) ncount = 1;
1327:          /* Fall through */
1328:
1329:        ANYNL02:
1330:        case 0x000a:
1331:          if (codevalue == OP_ANYNL_EXTRA + OP_TYPEPOSSTAR ||
1332:              codevalue == OP_ANYNL_EXTRA + OP_TYPEPOSQUERY)
1333:          {
1334:            active_count--;      /* Remove non-match possibility */
1335:            next_active_state--;
1336:          }
1337:          ADD_NEW_DATA(-(state_offset + count), 0, ncount);

```

```

1338:     break;
1339:
1340:     default:
1341:         break;
1342:     }
1343: }
1344: break;
1345:
1346: /*-----*/
1347: case OP_VSPACE_EXTRA + OP_TYPEQUERY:
1348: case OP_VSPACE_EXTRA + OP_TYPEMINQUERY:
1349: case OP_VSPACE_EXTRA + OP_TYPEPOSQUERY:
1350:     count = 2;
1351:     goto QS4;
1352:
1353: case OP_VSPACE_EXTRA + OP_TYPESTAR:
1354: case OP_VSPACE_EXTRA + OP_TYPEMINSTAR:
1355: case OP_VSPACE_EXTRA + OP_TYPEPOSSTAR:
1356:     count = 0;
1357:
1358: QS4:
1359:     ADD_ACTIVE(state_offset + 2, 0);
1360:     if (clen > 0)
1361:     {
1362:         BOOL OK;
1363:         switch (c)
1364:         {
1365:             case 0x000a:
1366:             case 0x000b:
1367:             case 0x000c:
1368:             case 0x000d:
1369:             case 0x0085:
1370:             case 0x2028:
1371:             case 0x2029:
1372:                 OK = TRUE;
1373:                 break;
1374:
1375:             default:
1376:                 OK = FALSE;
1377:                 break;
1378:         }

```

```

1379:     if (OK == (d == OP_VSPACE))
1380:     {
1381:         if (codevalue == OP_VSPACE_EXTRA + OP_TYPEPOSSTAR ||
1382:             codevalue == OP_VSPACE_EXTRA + OP_TYPEPOSQUERY)
1383:         {
1384:             active_count--;          /* Remove non-match possibility */
1385:             next_active_state--;
1386:         }
1387:         ADD_NEW_DATA(-(state_offset + count), 0, 0);
1388:     }
1389: }
1390: break;
1391:
1392: /*-----*/
1393: case OP_HSPACE_EXTRA + OP_TYPEQUERY:
1394: case OP_HSPACE_EXTRA + OP_TYPEMINQUERY:
1395: case OP_HSPACE_EXTRA + OP_TYPEPOSQUERY:
1396:     count = 2;
1397:     goto QS5;
1398:
1399: case OP_HSPACE_EXTRA + OP_TYPESTAR:
1400: case OP_HSPACE_EXTRA + OP_TYPEMINSTAR:
1401: case OP_HSPACE_EXTRA + OP_TYPEPOSSTAR:
1402:     count = 0;
1403:
1404: QS5:
1405:     ADD_ACTIVE(state_offset + 2, 0);
1406:     if (clen > 0)
1407:     {
1408:         BOOL OK;
1409:         switch (c)
1410:         {
1411:             case 0x09:    /* HT */
1412:             case 0x20:    /* SPACE */
1413:             case 0xa0:    /* NBSP */
1414:             case 0x1680:  /* OGHAM SPACE MARK */
1415:             case 0x180e:  /* MONGOLIAN VOWEL SEPARATOR */
1416:             case 0x2000:  /* EN QUAD */
1417:             case 0x2001:  /* EM QUAD */
1418:             case 0x2002:  /* EN SPACE */
1419:             case 0x2003:  /* EM SPACE */

```



```

1420:     case 0x2004: /* THREE-PER-EM SPACE */
1421:     case 0x2005: /* FOUR-PER-EM SPACE */
1422:     case 0x2006: /* SIX-PER-EM SPACE */
1423:     case 0x2007: /* FIGURE SPACE */
1424:     case 0x2008: /* PUNCTUATION SPACE */
1425:     case 0x2009: /* THIN SPACE */
1426:     case 0x200A: /* HAIR SPACE */
1427:     case 0x202f: /* NARROW NO-BREAK SPACE */
1428:     case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
1429:     case 0x3000: /* IDEOGRAPHIC SPACE */
1430:     OK = TRUE;
1431:     break;
1432:
1433:     default:
1434:     OK = FALSE;
1435:     break;
1436: }
1437:
1438: if (OK == (d == OP_HSPACE))
1439: {
1440:     if (codevalue == OP_HSPACE_EXTRA + OP_TYPEPOSTAR ||
1441:         codevalue == OP_HSPACE_EXTRA + OP_TYPEPOSTQUERY)
1442:     {
1443:         active_count--; /* Remove non-match possibility */
1444:         next_active_state--;
1445:     }
1446:     ADD_NEW_DATA(-(state_offset + count), 0, 0);
1447: }
1448: }
1449: break;
1450:
1451: /*-----*/
1452: #ifdef SUPPORT_UCP
1453:     case OP_PROP_EXTRA + OP_TYPEEXACT:
1454:     case OP_PROP_EXTRA + OP_TYPEUPTO:
1455:     case OP_PROP_EXTRA + OP_TYPEMINUPTO:
1456:     case OP_PROP_EXTRA + OP_TYPEPOSUPTO:
1457:     if (codevalue != OP_PROP_EXTRA + OP_TYPEEXACT)
1458:     { ADD_ACTIVE(state_offset + 6, 0); }
1459:     count = current_state->count; /* Number already matched */
1460:     if (clen > 0)

```

```

1461:     {
1462:     BOOL OK;
1463:     const ucd_record * prop = GET_UCD(c);
1464:     switch(code[4])
1465:     {
1466:     case PT_ANY:
1467:     OK = TRUE;
1468:     break;
1469:
1470:     case PT_LAMP:
1471:     OK = prop->chartype == ucp_Lu || prop->chartype == ucp_Ll || prop->chartype == ucp_Lt;
1472:     break;
1473:
1474:     case PT_GC:
1475:     OK = _pcre_ucp_gentype[prop->chartype] == code[5];
1476:     break;
1477:
1478:     case PT_PC:
1479:     OK = prop->chartype == code[5];
1480:     break;
1481:
1482:     case PT_SC:
1483:     OK = prop->script == code[5];
1484:     break;
1485:
1486:     /* Should never occur, but keep compilers from grumbling. */
1487:
1488:     default:
1489:     OK = codevalue != OP_PROP;
1490:     break;
1491:     }
1492:
1493:     if (OK == (d == OP_PROP))
1494:     {
1495:     if (codevalue == OP_PROP_EXTRA + OP_TYPEPOSUPTO)
1496:     {
1497:     active_count--;      /* Remove non-match possibility */
1498:     next_active_state--;
1499:     }
1500:     if (++count >= GET2(code, 1))
1501:     { ADD_NEW(state_offset + 6, 0); }

```

```

1502:     else
1503:         { ADD_NEW(state_offset, count); }
1504:     }
1505: }
1506: break;
1507:
1508: /*-----*/
1509: case OP_EXTUNI_EXTRA + OP_TYPEEXACT:
1510: case OP_EXTUNI_EXTRA + OP_TYPEUPTO:
1511: case OP_EXTUNI_EXTRA + OP_TYPEMINUPTO:
1512: case OP_EXTUNI_EXTRA + OP_TYPEPOSUPTO:
1513: if (codevalue != OP_EXTUNI_EXTRA + OP_TYPEEXACT)
1514:     { ADD_ACTIVE(state_offset + 4, 0); }
1515: count = current_state->count; /* Number already matched */
1516: if (clen > 0 && UCD_CATEGORY(c) != ucp_M)
1517:     {
1518:         const uschar *nptr = ptr + clen;
1519:         int ncount = 0;
1520:         if (codevalue == OP_EXTUNI_EXTRA + OP_TYPEPOSUPTO)
1521:             {
1522:                 active_count--; /* Remove non-match possibility */
1523:                 next_active_state--;
1524:             }
1525:         while (nptr < end_subject)
1526:             {
1527:                 int nd;
1528:                 int ndlen = 1;
1529:                 GETCHARLEN(nd, nptr, ndlen);
1530:                 if (UCD_CATEGORY(nd) != ucp_M) break;
1531:                 ncount++;
1532:                 nptr += ndlen;
1533:             }
1534:         if (++count >= GET2(code, 1))
1535:             { ADD_NEW_DATA(-(state_offset + 4), 0, ncount); }
1536:         else
1537:             { ADD_NEW_DATA(-state_offset, count, ncount); }
1538:     }
1539:     break;
1540: #endif
1541:
1542: /*-----*/

```

```

1543: case OP_ANYNL_EXTRA + OP_TYPEEEXACT:
1544: case OP_ANYNL_EXTRA + OP_TYPEUPTO:
1545: case OP_ANYNL_EXTRA + OP_TYPEMINUPTO:
1546: case OP_ANYNL_EXTRA + OP_TYPEPOSUPTO:
1547: if (codevalue != OP_ANYNL_EXTRA + OP_TYPEEEXACT)
1548:     { ADD_ACTIVE(state_offset + 4, 0); }
1549: count = current_state->count; /* Number already matched */
1550: if (clen > 0)
1551:     {
1552:         int ncount = 0;
1553:         switch (c)
1554:         {
1555:             case 0x000b:
1556:             case 0x000c:
1557:             case 0x0085:
1558:             case 0x2028:
1559:             case 0x2029:
1560:             if ((md->moptions & PCRE_BSR_ANYCRLF) != 0) break;
1561:             goto ANYNL03;
1562:
1563:             case 0x000d:
1564:             if (ptr + 1 < end_subject && ptr[1] == 0x0a) ncount = 1;
1565:             /* Fall through */
1566:
1567:             ANYNL03:
1568:             case 0x000a:
1569:             if (codevalue == OP_ANYNL_EXTRA + OP_TYPEPOSUPTO)
1570:                 {
1571:                     active_count--; /* Remove non-match possibility */
1572:                     next_active_state--;
1573:                 }
1574:             if (++count >= GET2(code, 1))
1575:                 { ADD_NEW_DATA(-(state_offset + 4), 0, ncount); }
1576:             else
1577:                 { ADD_NEW_DATA(-state_offset, count, ncount); }
1578:             break;
1579:
1580:             default:
1581:             break;
1582:         }
1583:     }

```

```

1584: break;
1585:
1586: /*-----*/
1587: case OP_VSPACE_EXTRA + OP_TYPEEXACT:
1588: case OP_VSPACE_EXTRA + OP_TYPEUPTO:
1589: case OP_VSPACE_EXTRA + OP_TYPEMINUPTO:
1590: case OP_VSPACE_EXTRA + OP_TYPEPOSUPTO:
1591: if (codevalue != OP_VSPACE_EXTRA + OP_TYPEEXACT)
1592:     { ADD_ACTIVE(state_offset + 4, 0); }
1593: count = current_state->count; /* Number already matched */
1594: if (klen > 0)
1595:     {
1596:     BOOL OK;
1597:     switch (c)
1598:     {
1599:     case 0x000a:
1600:     case 0x000b:
1601:     case 0x000c:
1602:     case 0x000d:
1603:     case 0x0085:
1604:     case 0x2028:
1605:     case 0x2029:
1606:     OK = TRUE;
1607:     break;
1608:
1609:     default:
1610:     OK = FALSE;
1611:     }
1612:
1613: if (OK == (d == OP_VSPACE))
1614:     {
1615:     if (codevalue == OP_VSPACE_EXTRA + OP_TYPEPOSUPTO)
1616:     {
1617:     active_count--; /* Remove non-match possibility */
1618:     next_active_state--;
1619:     }
1620:     if (++count >= GET2(code, 1))
1621:     { ADD_NEW_DATA(-(state_offset + 4), 0, 0); }
1622:     else
1623:     { ADD_NEW_DATA(-state_offset, count, 0); }
1624:     }

```

```

1625:     }
1626: break;
1627:
1628: /*-----*/
1629: case OP_HSPACE_EXTRA + OP_TYPEEEXACT:
1630: case OP_HSPACE_EXTRA + OP_TYPEUPTO:
1631: case OP_HSPACE_EXTRA + OP_TYPEMINUPTO:
1632: case OP_HSPACE_EXTRA + OP_TYPEPOSUPTO:
1633: if (codevalue != OP_HSPACE_EXTRA + OP_TYPEEEXACT)
1634:     { ADD_ACTIVE(state_offset + 4, 0); }
1635: count = current_state->count; /* Number already matched */
1636: if (clen > 0)
1637:     {
1638:     BOOL OK;
1639:     switch (c)
1640:     {
1641:     case 0x09: /* HT */
1642:     case 0x20: /* SPACE */
1643:     case 0xa0: /* NBSP */
1644:     case 0x1680: /* OGHAM SPACE MARK */
1645:     case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
1646:     case 0x2000: /* EN QUAD */
1647:     case 0x2001: /* EM QUAD */
1648:     case 0x2002: /* EN SPACE */
1649:     case 0x2003: /* EM SPACE */
1650:     case 0x2004: /* THREE-PER-EM SPACE */
1651:     case 0x2005: /* FOUR-PER-EM SPACE */
1652:     case 0x2006: /* SIX-PER-EM SPACE */
1653:     case 0x2007: /* FIGURE SPACE */
1654:     case 0x2008: /* PUNCTUATION SPACE */
1655:     case 0x2009: /* THIN SPACE */
1656:     case 0x200A: /* HAIR SPACE */
1657:     case 0x202f: /* NARROW NO-BREAK SPACE */
1658:     case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
1659:     case 0x3000: /* IDEOGRAPHIC SPACE */
1660:     OK = TRUE;
1661:     break;
1662:
1663:     default:
1664:     OK = FALSE;
1665:     break;

```

```

1666:     }
1667:
1668:     if (OK == (d == OP_HSPACE))
1669:     {
1670:         if (codevalue == OP_HSPACE_EXTRA + OP_TYPEPOSUPTO)
1671:         {
1672:             active_count--;      /* Remove non-match possibility */
1673:             next_active_state--;
1674:         }
1675:         if (++count >= GET2(code, 1))
1676:             { ADD_NEW_DATA(-(state_offset + 4), 0, 0); }
1677:         else
1678:             { ADD_NEW_DATA(-state_offset, count, 0); }
1679:     }
1680: }
1681: break;
1682:
1683:                                     /*
=====
*/
1684: /* These opcodes are followed by a character that is usually compared
1685: to the current subject character; it is loaded into d. We still get
1686: here even if there is no subject character, because in some cases zero
1687: repetitions are permitted. */
1688:
1689: /*-----*/
1690: case OP_CHAR:
1691:     if (clen > 0 && c == d) { ADD_NEW(state_offset + dlen + 1, 0); }
1692:     break;
1693:
1694: /*-----*/
1695: case OP_CHARNC:
1696:     if (clen == 0) break;
1697:
1698: #ifdef SUPPORT_UTF8
1699:     if (utf8)
1700:     {
1701:         if (c == d) { ADD_NEW(state_offset + dlen + 1, 0); } else
1702:         {
1703:             unsigned int othercase;
1704:             if (c < 128) othercase = fcc[c]; else

```

```

1705:
1706:     /* If we have Unicode property support, we can use it to test the
1707:        other case of the character. */
1708:
1709: #ifdef SUPPORT_UCP
1710:     othercase = UCD_OTHERCASE(c);
1711: #else
1712:     othercase = NOTACHAR;
1713: #endif
1714:
1715:     if (d == othercase) { ADD_NEW(state_offset + dlen + 1, 0); }
1716:     }
1717:     }
1718:     else
1719: #endif /* SUPPORT_UTF8 */
1720:
1721:     /* Non-UTF-8 mode */
1722:     {
1723:         if (lcc[c] == lcc[d]) { ADD_NEW(state_offset + 2, 0); }
1724:     }
1725:     break;
1726:
1727:
1728: #ifdef SUPPORT_UCP
1729:     /*-----*/
1730:     /* This is a tricky one because it can match more than one character.
1731:        Find out how many characters to skip, and then set up a negative state
1732:        to wait for them to pass before continuing. */
1733:
1734:     case OP_EXTUNI:
1735:         if (clen > 0 && UCD_CATEGORY(c) != ucp_M)
1736:         {
1737:             const uschar *nptr = ptr + clen;
1738:             int ncount = 0;
1739:             while (nptr < end_subject)
1740:             {
1741:                 int nclen = 1;
1742:                 GETCHARLEN(c, nptr, nclen);
1743:                 if (UCD_CATEGORY(c) != ucp_M) break;
1744:                 ncount++;
1745:                 nptr += nclen;

```



```

1746:     }
1747:     ADD_NEW_DATA(-(state_offset + 1), 0, ncount);
1748:     }
1749:     break;
1750: #endif
1751:
1752: /*-----*/
1753: /* This is a tricky like EXTUNI because it too can match more than one
1754: character (when CR is followed by LF). In this case, set up a negative
1755: state to wait for one character to pass before continuing. */
1756:
1757: case OP_ANYNL:
1758: if (clen > 0) switch(c)
1759: {
1760:     case 0x000b:
1761:     case 0x000c:
1762:     case 0x0085:
1763:     case 0x2028:
1764:     case 0x2029:
1765:     if ((md->moptions & PCRE_BSR_ANYCRLF) != 0) break;
1766:
1767:     case 0x000a:
1768:     ADD_NEW(state_offset + 1, 0);
1769:     break;
1770:
1771:     case 0x000d:
1772:     if (ptr + 1 < end_subject && ptr[1] == 0x0a)
1773:     {
1774:     ADD_NEW_DATA(-(state_offset + 1), 0, 1);
1775:     }
1776:     else
1777:     {
1778:     ADD_NEW(state_offset + 1, 0);
1779:     }
1780:     break;
1781:     }
1782: break;
1783:
1784: /*-----*/
1785: case OP_NOT_VSPACE:
1786: if (clen > 0) switch(c)

```

```

1787:     {
1788:     case 0x000a:
1789:     case 0x000b:
1790:     case 0x000c:
1791:     case 0x000d:
1792:     case 0x0085:
1793:     case 0x2028:
1794:     case 0x2029:
1795:     break;
1796:
1797:     default:
1798:     ADD_NEW(state_offset + 1, 0);
1799:     break;
1800:     }
1801: break;
1802:
1803: /*-----*/
1804: case OP_VSPACE:
1805: if (clen > 0) switch(c)
1806: {
1807:     case 0x000a:
1808:     case 0x000b:
1809:     case 0x000c:
1810:     case 0x000d:
1811:     case 0x0085:
1812:     case 0x2028:
1813:     case 0x2029:
1814:     ADD_NEW(state_offset + 1, 0);
1815:     break;
1816:
1817:     default: break;
1818: }
1819: break;
1820:
1821: /*-----*/
1822: case OP_NOT_HSPACE:
1823: if (clen > 0) switch(c)
1824: {
1825:     case 0x09:    /* HT */
1826:     case 0x20:    /* SPACE */
1827:     case 0xa0:    /* NBSP */

```

```

1828:     case 0x1680: /* OGHAM SPACE MARK */
1829:     case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
1830:     case 0x2000: /* EN QUAD */
1831:     case 0x2001: /* EM QUAD */
1832:     case 0x2002: /* EN SPACE */
1833:     case 0x2003: /* EM SPACE */
1834:     case 0x2004: /* THREE-PER-EM SPACE */
1835:     case 0x2005: /* FOUR-PER-EM SPACE */
1836:     case 0x2006: /* SIX-PER-EM SPACE */
1837:     case 0x2007: /* FIGURE SPACE */
1838:     case 0x2008: /* PUNCTUATION SPACE */
1839:     case 0x2009: /* THIN SPACE */
1840:     case 0x200A: /* HAIR SPACE */
1841:     case 0x202f: /* NARROW NO-BREAK SPACE */
1842:     case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
1843:     case 0x3000: /* IDEOGRAPHIC SPACE */
1844:     break;
1845:
1846:     default:
1847:         ADD_NEW(state_offset + 1, 0);
1848:     break;
1849: }
1850: break;
1851:
1852: /*-----*/
1853: case OP_HSPACE:
1854: if (clen > 0) switch(c)
1855: {
1856:     case 0x09: /* HT */
1857:     case 0x20: /* SPACE */
1858:     case 0xa0: /* NBSP */
1859:     case 0x1680: /* OGHAM SPACE MARK */
1860:     case 0x180e: /* MONGOLIAN VOWEL SEPARATOR */
1861:     case 0x2000: /* EN QUAD */
1862:     case 0x2001: /* EM QUAD */
1863:     case 0x2002: /* EN SPACE */
1864:     case 0x2003: /* EM SPACE */
1865:     case 0x2004: /* THREE-PER-EM SPACE */
1866:     case 0x2005: /* FOUR-PER-EM SPACE */
1867:     case 0x2006: /* SIX-PER-EM SPACE */
1868:     case 0x2007: /* FIGURE SPACE */

```

```

1869:     case 0x2008: /* PUNCTUATION SPACE */
1870:     case 0x2009: /* THIN SPACE */
1871:     case 0x200A: /* HAIR SPACE */
1872:     case 0x202f: /* NARROW NO-BREAK SPACE */
1873:     case 0x205f: /* MEDIUM MATHEMATICAL SPACE */
1874:     case 0x3000: /* IDEOGRAPHIC SPACE */
1875:     ADD_NEW(state_offset + 1, 0);
1876:     break;
1877: }
1878: break;
1879:
1880: /*-----*/
1881: /* Match a negated single character. This is only used for one-byte
1882: characters, that is, we know that d < 256. The character we are
1883: checking (c) can be multibyte. */
1884:
1885: case OP_NOT:
1886:     if (klen > 0)
1887:     {
1888:         unsigned int otherd = ((ims & PCRE_CASELESS) != 0)? fcc[d] : d;
1889:         if (c != d && c != otherd) { ADD_NEW(state_offset + dlen + 1, 0); }
1890:     }
1891:     break;
1892:
1893: /*-----*/
1894: case OP_PLUS:
1895: case OP_MINPLUS:
1896: case OP_POSPLUS:
1897: case OP_NOTPLUS:
1898: case OP_NOTMINPLUS:
1899: case OP_NOTPOSPLUS:
1900:     count = current_state->count; /* Already matched */
1901:     if (count > 0) { ADD_ACTIVE(state_offset + dlen + 1, 0); }
1902:     if (klen > 0)
1903:     {
1904:         unsigned int otherd = NOTACHAR;
1905:         if ((ims & PCRE_CASELESS) != 0)
1906:         {
1907: #ifdef SUPPORT_UTF8
1908:             if (utf8 && d >= 128)
1909:             {

```

```

1910: #ifdef SUPPORT_UCP
1911:     otherd = UCD_OTHERCASE(d);
1912: #endif /* SUPPORT_UCP */
1913:     }
1914:     else
1915: #endif /* SUPPORT_UTF8 */
1916:     otherd = fcc[d];
1917:     }
1918:     if ((c == d || c == otherd) == (codevalue < OP_NOTSTAR))
1919:     {
1920:         if (count > 0 &&
1921:             (codevalue == OP_POSPLUS || codevalue == OP_NOTPOSPLUS))
1922:         {
1923:             active_count--; /* Remove non-match possibility */
1924:             next_active_state--;
1925:         }
1926:         count++;
1927:         ADD_NEW(state_offset, count);
1928:     }
1929: }
1930: break;
1931:
1932: /*-----*/
1933: case OP_QUERY:
1934: case OP_MINQUERY:
1935: case OP_POSQUERY:
1936: case OP_NOTQUERY:
1937: case OP_NOTMINQUERY:
1938: case OP_NOTPOSQUERY:
1939:     ADD_ACTIVE(state_offset + clen + 1, 0);
1940:     if (clen > 0)
1941:     {
1942:         unsigned int otherd = NOTACHAR;
1943:         if ((ims & PCRE_CASELESS) != 0)
1944:         {
1945: #ifdef SUPPORT_UTF8
1946:             if (utf8 && d >= 128)
1947:             {
1948: #ifdef SUPPORT_UCP
1949:                 otherd = UCD_OTHERCASE(d);
1950: #endif /* SUPPORT_UCP */

```

```

1951:     }
1952:     else
1953: #endif /* SUPPORT_UTF8 */
1954:     otherd = fcc[d];
1955:     }
1956:     if ((c == d || c == otherd) == (codevalue < OP_NOTSTAR))
1957:     {
1958:         if (codevalue == OP_POSQUERY || codevalue == OP_NOTPOSQUERY)
1959:         {
1960:             active_count--;          /* Remove non-match possibility */
1961:             next_active_state--;
1962:         }
1963:         ADD_NEW(state_offset + dlen + 1, 0);
1964:     }
1965: }
1966: break;
1967:
1968: /*-----*/
1969: case OP_STAR:
1970: case OP_MINSTAR:
1971: case OP_POSSTAR:
1972: case OP_NOTSTAR:
1973: case OP_NOTMINSTAR:
1974: case OP_NOTPOSSTAR:
1975:     ADD_ACTIVE(state_offset + dlen + 1, 0);
1976:     if (clen > 0)
1977:     {
1978:         unsigned int otherd = NOTACHAR;
1979:         if ((ims & PCRE_CASELESS) != 0)
1980:         {
1981: #ifdef SUPPORT_UTF8
1982:             if (utf8 && d >= 128)
1983:             {
1984: #ifdef SUPPORT_UCP
1985:                 otherd = UCD_OTHERCASE(d);
1986: #endif /* SUPPORT_UCP */
1987:             }
1988:         else
1989: #endif /* SUPPORT_UTF8 */
1990:             otherd = fcc[d];
1991:         }

```

```

1992:     if ((c == d || c == otherd) == (codevalue < OP_NOTSTAR))
1993:     {
1994:         if (codevalue == OP_POSSTAR || codevalue == OP_NOTPOSSTAR)
1995:         {
1996:             active_count--;          /* Remove non-match possibility */
1997:             next_active_state--;
1998:         }
1999:         ADD_NEW(state_offset, 0);
2000:     }
2001: }
2002: break;
2003:
2004: /*-----*/
2005: case OP_EXACT:
2006: case OP_NOTEXACT:
2007:     count = current_state->count; /* Number already matched */
2008:     if (clen > 0)
2009:     {
2010:         unsigned int otherd = NOTACHAR;
2011:         if ((ims & PCRE_CASELESS) != 0)
2012:         {
2013: #ifdef SUPPORT_UTF8
2014:             if (utf8 && d >= 128)
2015:             {
2016: #ifdef SUPPORT_UCP
2017:                 otherd = UCD_OTHERCASE(d);
2018: #endif /* SUPPORT_UCP */
2019:             }
2020:             else
2021: #endif /* SUPPORT_UTF8 */
2022:                 otherd = fcc[d];
2023:         }
2024:         if ((c == d || c == otherd) == (codevalue < OP_NOTSTAR))
2025:         {
2026:             if (++count >= GET2(code, 1))
2027:                 { ADD_NEW(state_offset + dlen + 3, 0); }
2028:             else
2029:                 { ADD_NEW(state_offset, count); }
2030:         }
2031:     }
2032:     break;

```

```

2033:
2034:  /*-----*/
2035:  case OP_UPTO:
2036:  case OP_MINUPTO:
2037:  case OP_POSUPTO:
2038:  case OP_NOTUPTO:
2039:  case OP_NOTMINUPTO:
2040:  case OP_NOTPOSUPTO:
2041:  ADD_ACTIVE(state_offset + dlen + 3, 0);
2042:  count = current_state->count; /* Number already matched */
2043:  if (klen > 0)
2044:  {
2045:      unsigned int otherd = NOTACHAR;
2046:      if ((ims & PCRE_CASELESS) != 0)
2047:      {
2048: #ifdef SUPPORT_UTF8
2049:         if (utf8 && d >= 128)
2050:         {
2051: #ifdef SUPPORT_UCP
2052:            otherd = UCD_OTHERCASE(d);
2053: #endif /* SUPPORT_UCP */
2054:         }
2055:         else
2056: #endif /* SUPPORT_UTF8 */
2057:            otherd = fcc[d];
2058:      }
2059:      if ((c == d || c == otherd) == (codevalue < OP_NOTSTAR))
2060:      {
2061:         if (codevalue == OP_POSUPTO || codevalue == OP_NOTPOSUPTO)
2062:         {
2063:             active_count--; /* Remove non-match possibility */
2064:             next_active_state--;
2065:         }
2066:         if (++count >= GET2(code, 1))
2067:             { ADD_NEW(state_offset + dlen + 3, 0); }
2068:         else
2069:             { ADD_NEW(state_offset, count); }
2070:      }
2071:  }
2072:  break;
2073:

```



```

2074:
2075:
=====
2076:  /* These are the class-handling opcodes */
2077:
2078:  case OP_CLASS:
2079:  case OP_NCLASS:
2080:  case OP_XCLASS:
2081:  {
2082:      BOOL isinclass = FALSE;
2083:      int next_state_offset;
2084:      const uschar *ecode;
2085:
2086:      /* For a simple class, there is always just a 32-byte table, and we
2087:       can set isinclass from it. */
2088:
2089:      if (codevalue != OP_XCLASS)
2090:      {
2091:          ecode = code + 33;
2092:          if (clen > 0)
2093:          {
2094:              isinclass = (c > 255)? (codevalue == OP_NCLASS) :
2095:                  ((code[1 + c/8] & (1 << (c&7))) != 0);
2096:          }
2097:      }
2098:
2099:      /* An extended class may have a table or a list of single characters,
2100:       ranges, or both, and it may be positive or negative. There's a
2101:       function that sorts all this out. */
2102:
2103:      else
2104:      {
2105:          ecode = code + GET(code, 1);
2106:          if (clen > 0) isinclass = _pcre_xclass(c, code + 1 + LINK_SIZE);
2107:      }
2108:
2109:      /* At this point, isinclass is set for all kinds of class, and ecode
2110:       points to the byte after the end of the class. If there is a
2111:       quantifier, this is where it will be. */
2112:

```

```

2113:     next_state_offset = ecode - start_code;
2114:
2115:     switch (*ecode)
2116:     {
2117:     case OP_CRSTAR:
2118:     case OP_CRMINSTAR:
2119:         ADD_ACTIVE(next_state_offset + 1, 0);
2120:         if (isinclass) { ADD_NEW(state_offset, 0); }
2121:         break;
2122:
2123:     case OP_CRPLUS:
2124:     case OP_CRMINPLUS:
2125:         count = current_state->count; /* Already matched */
2126:         if (count > 0) { ADD_ACTIVE(next_state_offset + 1, 0); }
2127:         if (isinclass) { count++; ADD_NEW(state_offset, count); }
2128:         break;
2129:
2130:     case OP_CRQUERY:
2131:     case OP_CRMINQUERY:
2132:         ADD_ACTIVE(next_state_offset + 1, 0);
2133:         if (isinclass) { ADD_NEW(next_state_offset + 1, 0); }
2134:         break;
2135:
2136:     case OP_CRRANGE:
2137:     case OP_CRMINRANGE:
2138:         count = current_state->count; /* Already matched */
2139:         if (count >= GET2(ecode, 1))
2140:             { ADD_ACTIVE(next_state_offset + 5, 0); }
2141:         if (isinclass)
2142:             {
2143:                 int max = GET2(ecode, 3);
2144:                 if (++count >= max && max != 0) /* Max 0 => no limit */
2145:                     { ADD_NEW(next_state_offset + 5, 0); }
2146:                 else
2147:                     { ADD_NEW(state_offset, count); }
2148:             }
2149:         break;
2150:
2151:     default:
2152:         if (isinclass) { ADD_NEW(next_state_offset, 0); }
2153:         break;

```

```

2154:     }
2155: }
2156: break;
2157:
2158:                                     /*
=====
*/
2159: /* These are the opcodes for fancy brackets of various kinds. We have
2160: to use recursion in order to handle them. The "always failing" assertion
2161: (!?) is optimised when compiling to OP_FAIL, so we have to support that,
2162: though the other "backtracking verbs" are not supported. */
2163:
2164: case OP_FAIL:
2165: break;
2166:
2167: case OP_ASSERT:
2168: case OP_ASSERT_NOT:
2169: case OP_ASSERTBACK:
2170: case OP_ASSERTBACK_NOT:
2171: {
2172: int rc;
2173: int local_offsets[2];
2174: int local_workspace[1000];
2175: const uschar *endasscode = code + GET(code, 1);
2176:
2177: while (*endasscode == OP_ALT) endasscode += GET(endasscode, 1);
2178:
2179: rc = internal_dfa_exec(
2180:     md,                /* static match data */
2181:     code,               /* this subexpression's code */
2182:     ptr,                /* where we currently are */
2183:     ptr - start_subject, /* start offset */
2184:     local_offsets,      /* offset vector */
2185:     sizeof(local_offsets)/sizeof(int), /* size of same */
2186:     local_workspace,    /* workspace vector */
2187:     sizeof(local_workspace)/sizeof(int), /* size of same */
2188:     ims,                /* the current ims flags */
2189:     rlevel,             /* function recursion level */
2190:     recursing);         /* pass on regex recursion */
2191:
2192: if ((rc >= 0) == (codevalue == OP_ASSERT || codevalue == OP_ASSERTBACK))

```

```

2193:     { ADD_ACTIVE(endasscode + LINK_SIZE + 1 - start_code, 0); }
2194: }
2195: break;
2196:
2197: /*-----*/
2198: case OP_COND:
2199: case OP_SCOND:
2200: {
2201:     int local_offsets[1000];
2202:     int local_workspace[1000];
2203:     int conddcode = code[LINK_SIZE+1];
2204:
2205:     /* Back reference conditions are not supported */
2206:
2207:     if (conddcode == OP_CREF) return PCRE_ERROR_DFA_UCOND;
2208:
2209:     /* The DEFINE condition is always false */
2210:
2211:     if (conddcode == OP_DEF)
2212:     {
2213:         ADD_ACTIVE(state_offset + GET(code, 1) + LINK_SIZE + 1, 0);
2214:     }
2215:
2216:     /* The only supported version of OP_RREF is for the value RREF_ANY,
2217:     which means "test if in any recursion". We can't test for specifically
2218:     recursed groups. */
2219:
2220:     else if (conddcode == OP_RREF)
2221:     {
2222:         int value = GET2(code, LINK_SIZE+2);
2223:         if (value != RREF_ANY) return PCRE_ERROR_DFA_UCOND;
2224:         if (recurring > 0) { ADD_ACTIVE(state_offset + LINK_SIZE + 4, 0); }
2225:         else { ADD_ACTIVE(state_offset + GET(code, 1) + LINK_SIZE + 1, 0); }
2226:     }
2227:
2228:     /* Otherwise, the condition is an assertion */
2229:
2230:     else
2231:     {
2232:         int rc;
2233:         const uschar *asscode = code + LINK_SIZE + 1;

```

```

2234:     const uschar *endasscode = asscode + GET(asscode, 1);
2235:
2236:     while (*endasscode == OP_ALT) endasscode += GET(endasscode, 1);
2237:
2238:     rc = internal_dfa_exec(
2239:         md,                /* fixed match data */
2240:         asscode,           /* this subexpression's code */
2241:         ptr,               /* where we currently are */
2242:         ptr - start_subject, /* start offset */
2243:         local_offsets,     /* offset vector */
2244:         sizeof(local_offsets)/sizeof(int), /* size of same */
2245:         local_workspace,   /* workspace vector */
2246:         sizeof(local_workspace)/sizeof(int), /* size of same */
2247:         ims,               /* the current ims flags */
2248:         rlevel,            /* function recursion level */
2249:         recursing);        /* pass on regex recursion */
2250:
2251:     if ((rc >= 0) ==
2252:         (condcode == OP_ASSERT || condcode == OP_ASSERTBACK))
2253:         { ADD_ACTIVE(endasscode + LINK_SIZE + 1 - start_code, 0); }
2254:     else
2255:         { ADD_ACTIVE(state_offset + GET(code, 1) + LINK_SIZE + 1, 0); }
2256:     }
2257: }
2258: break;
2259:
2260: /*-----*/
2261: case OP_RECURSE:
2262:     {
2263:         int local_offsets[1000];
2264:         int local_workspace[1000];
2265:         int rc;
2266:
2267:         DPRINTF(("%. *sStarting regex recursion %d \n", rlevel*2-2, SP,
2268:             recursing + 1));
2269:
2270:         rc = internal_dfa_exec(
2271:             md,                /* fixed match data */
2272:             start_code + GET(code, 1), /* this subexpression's code */
2273:             ptr,               /* where we currently are */
2274:             ptr - start_subject, /* start offset */

```

```

2275:     local_offsets,                /* offset vector */
2276:     sizeof(local_offsets)/sizeof(int), /* size of same */
2277:     local_workspace,              /* workspace vector */
2278:     sizeof(local_workspace)/sizeof(int), /* size of same */
2279:     ims,                          /* the current ims flags */
2280:     rlevel,                       /* function recursion level */
2281:     recursing + 1);               /* regex recurse level */
2282:
2283:     DPRINTF(("%.sReturn from regex recursion %d: rc=%d \n", rlevel*2-2, SP,
2284:     recursing + 1, rc));
2285:
2286:     /* Ran out of internal offsets */
2287:
2288:     if (rc == 0) return PCRE_ERROR_DFA_RECURSE;
2289:
2290:     /* For each successful matched substring, set up the next state with a
2291:     count of characters to skip before trying it. Note that the count is in
2292:     characters, not bytes. */
2293:
2294:     if (rc > 0)
2295:     {
2296:         for (rc = rc*2 - 2; rc >= 0; rc -= 2)
2297:         {
2298:             const uschar *p = start_subject + local_offsets[rc];
2299:             const uschar *pp = start_subject + local_offsets[rc+1];
2300:             int charcount = local_offsets[rc+1] - local_offsets[rc];
2301:             while (p < pp) if ((*p++ & 0xc0) == 0x80) charcount--;
2302:             if (charcount > 0)
2303:             {
2304:                 ADD_NEW_DATA(-(state_offset + LINK_SIZE + 1), 0, (charcount - 1));
2305:             }
2306:             else
2307:             {
2308:                 ADD_ACTIVE(state_offset + LINK_SIZE + 1, 0);
2309:             }
2310:         }
2311:     }
2312:     else if (rc != PCRE_ERROR_NOMATCH) return rc;
2313: }
2314: break;
2315:

```

```

2316:  /*-----*/
2317:  case OP_ONCE:
2318:  {
2319:      int local_offsets[2];
2320:      int local_workspace[1000];
2321:
2322:      int rc = internal_dfa_exec(
2323:          md,                /* fixed match data */
2324:          code,              /* this subexpression's code */
2325:          ptr,               /* where we currently are */
2326:          ptr - start_subject, /* start offset */
2327:          local_offsets,      /* offset vector */
2328:          sizeof(local_offsets)/sizeof(int), /* size of same */
2329:          local_workspace,    /* workspace vector */
2330:          sizeof(local_workspace)/sizeof(int), /* size of same */
2331:          ims,                /* the current ims flags */
2332:          rlevel,             /* function recursion level */
2333:          recursing);         /* pass on regex recursion */
2334:
2335:      if (rc >= 0)
2336:      {
2337:          const uschar *end_subpattern = code;
2338:          int charcount = local_offsets[1] - local_offsets[0];
2339:          int next_state_offset, repeat_state_offset;
2340:
2341:          do { end_subpattern += GET(end_subpattern, 1); }
2342:              while (*end_subpattern == OP_ALT);
2343:          next_state_offset = end_subpattern - start_code + LINK_SIZE + 1;
2344:
2345:          /* If the end of this subpattern is KETRMAX or KETRMIN, we must
2346:             arrange for the repeat state also to be added to the relevant list.
2347:             Calculate the offset, or set -1 for no repeat. */
2348:
2349:          repeat_state_offset = (*end_subpattern == OP_KETRMAX ||
2350:                                *end_subpattern == OP_KETRMIN)?
2351:              end_subpattern - start_code - GET(end_subpattern, 1) : -1;
2352:
2353:          /* If we have matched an empty string, add the next state at the
2354:             current character pointer. This is important so that the duplicate
2355:             checking kicks in, which is what breaks infinite loops that match an
2356:             empty string. */

```

```

2357:
2358:     if (charcount == 0)
2359:     {
2360:         ADD_ACTIVE(next_state_offset, 0);
2361:     }
2362:
2363:     /* Optimization: if there are no more active states, and there
2364:     are no new states yet set up, then skip over the subject string
2365:     right here, to save looping. Otherwise, set up the new state to swing
2366:     into action when the end of the substring is reached. */
2367:
2368:     else if (i + 1 >= active_count && new_count == 0)
2369:     {
2370:         ptr += charcount;
2371:         clen = 0;
2372:         ADD_NEW(next_state_offset, 0);
2373:
2374:         /* If we are adding a repeat state at the new character position,
2375:         we must fudge things so that it is the only current state.
2376:         Otherwise, it might be a duplicate of one we processed before, and
2377:         that would cause it to be skipped. */
2378:
2379:         if (repeat_state_offset >= 0)
2380:         {
2381:             next_active_state = active_states;
2382:             active_count = 0;
2383:             i = -1;
2384:             ADD_ACTIVE(repeat_state_offset, 0);
2385:         }
2386:     }
2387:     else
2388:     {
2389:         const uschar *p = start_subject + local_offsets[0];
2390:         const uschar *pp = start_subject + local_offsets[1];
2391:         while (p < pp) if ((*p++ & 0xc0) == 0x80) charcount--;
2392:         ADD_NEW_DATA(-next_state_offset, 0, (charcount - 1));
2393:         if (repeat_state_offset >= 0)
2394:             { ADD_NEW_DATA(-repeat_state_offset, 0, (charcount - 1)); }
2395:     }
2396:
2397: }

```



```

2398:     else if (rc != PCRE_ERROR_NOMATCH) return rc;
2399:     }
2400:     break;
2401:
2402:
2403:
2404:
2405:
2406:
2407:
2408:
2409:
2410:
2411:
2412:
2413:
2414:
2415:
2416:
2417:
2418:
2419:
2420:
2421:
2422:
2423:
2424:
2425:
2426:
2427:
2428:
2429:
2430:
2431:
2432:
2433:
2434:
2435:

```

/*
 =====
 */
 /* Handle callouts */
 case OP_CALLOUT:
 if (pcre_callout != NULL)
 {
 int rrc;
 pcre_callout_block cb;
 cb.version = 1; /* Version 1 of the callout block */
 cb.callout_number = code[1];
 cb.offset_vector = offsets;
 cb.subject = (PCRE_SPTR)start_subject;
 cb.subject_length = end_subject - start_subject;
 cb.start_match = current_subject - start_subject;
 cb.current_position = ptr - start_subject;
 cb.pattern_position = GET(code, 2);
 cb.next_item_length = GET(code, 2 + LINK_SIZE);
 cb.capture_top = 1;
 cb.capture_last = -1;
 cb.callout_data = md->callout_data;
 if ((rrc = (*pcre_callout)(&cb)) < 0) return rrc; /* Abandon */
 if (rrc == 0) { ADD_ACTIVE(state_offset + 2 + 2*LINK_SIZE, 0); }
 }
 break;

/*
 =====
 */
 default: /* Unsupported opcode */
 return PCRE_ERROR_DFA_UITEM;
 }
 NEXT_ACTIVE_STATE: continue;

```

2436: }    /* End of loop scanning active states */
2437:
2438: /* We have finished the processing at the current subject character. If no
2439: new states have been set for the next character, we have found all the
2440: matches that we are going to find. If we are at the top level and partial
2441: matching has been requested, check for appropriate conditions. */
2442:
2443: if (new_count <= 0)
2444: {
2445:     if (match_count < 0 &&                /* No matches found */
2446:         rlevel == 1 &&                    /* Top level match function */
2447:         (md->moptions & PCRE_PARTIAL) != 0 && /* Want partial matching */
2448:         ptr >= end_subject &&             /* Reached end of subject */
2449:         ptr > current_subject)            /* Matched non-empty string */
2450:     {
2451:         if (offsetcount >= 2)
2452:         {
2453:             offsets[0] = current_subject - start_subject;
2454:             offsets[1] = end_subject - start_subject;
2455:         }
2456:         match_count = PCRE_ERROR_PARTIAL;
2457:     }
2458:
2459:     DPRINTF(("%.sEnd of internal_dfa_exec %d: returning %d \n"
2460:         "%.s----- \n \n", rlevel*2-2, SP, rlevel, match_count,
2461:         rlevel*2-2, SP));
2462:     break;    /* In effect, "return", but see the comment below */
2463: }
2464:
2465: /* One or more states are active for the next character. */
2466:
2467: ptr += clen; /* Advance to next subject character */
2468: }           /* Loop to move along the subject string */
2469:
2470: /* Control gets here from "break" a few lines above. We do it this way because
2471: if we use "return" above, we have compiler trouble. Some compilers warn if
2472: there's nothing here because they think the function doesn't return a value. On
2473: the other hand, if we put a dummy statement here, some more clever compilers
2474: complain that it can't be reached. Sigh. */
2475:
2476: return match_count;

```

```

2477: }
2478:
2479:
2480:
2481:
2482: /*****
2483: *   Execute a Regular Expression - DFA engine   *
2484: *****/
2485:
2486: /* This external function applies a compiled re to a subject string using a DFA
2487: engine. This function calls the internal function multiple times if the pattern
2488: is not anchored.
2489:
2490: Arguments:
2491: argument_re    points to the compiled expression
2492: extra_data     points to extra data or is NULL
2493: subject        points to the subject string
2494: length         length of subject string (may contain binary zeros)
2495: start_offset   where to start in the subject string
2496: options        option bits
2497: offsets        vector of match offsets
2498: offsetcount    size of same
2499: workspace      workspace vector
2500: wscount        size of same
2501:
2502: Returns:       > 0 => number of match offset pairs placed in offsets
2503:               = 0 => offsets overflowed; longest matches are present
2504:               -1 => failed to match
2505:               < -1 => some kind of unexpected problem
2506: */
2507:
2508: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
2509: pcre_dfa_exec(const pcre *argument_re, const pcre_extra *extra_data,
2510: const char *subject, int length, int start_offset, int options, int *offsets,
2511: int offsetcount, int *workspace, int wscount)
2512: {
2513: real_pcre *re = (real_pcre *)argument_re;
2514: dfa_match_data match_block;
2515: dfa_match_data *md = &match_block;
2516: BOOL utf8, anchored, startline, firstline;
2517: const uschar *current_subject, *end_subject, *lcc;

```

```

2518:
2519: pcre_study_data internal_study;
2520: const pcre_study_data *study = NULL;
2521: real_pcre internal_re;
2522:
2523: const uschar *req_byte_ptr;
2524: const uschar *start_bits = NULL;
2525: BOOL first_byte_caseless = FALSE;
2526: BOOL req_byte_caseless = FALSE;
2527: int first_byte = -1;
2528: int req_byte = -1;
2529: int req_byte2 = -1;
2530: int newline;
2531:
2532: /* Plausibility checks */
2533:
2534: if ((options & ~PUBLIC_DFA_EXEC_OPTIONS) != 0) return PCRE_ERROR_BADOPTION;
2535: if (re == NULL || subject == NULL || workspace == NULL ||
2536:     (offsets == NULL && offsetcount > 0)) return PCRE_ERROR_NULL;
2537: if (offsetcount < 0) return PCRE_ERROR_BADCOUNT;
2538: if (wscount < 20) return PCRE_ERROR_DFA_WSSIZE;
2539:
2540: /* We need to find the pointer to any study data before we test for byte
2541:  flipping, so we scan the extra_data block first. This may set two fields in the
2542:  match block, so we must initialize them beforehand. However, the other fields
2543:  in the match block must not be set until after the byte flipping. */
2544:
2545: md->tables = re->tables;
2546: md->callout_data = NULL;
2547:
2548: if (extra_data != NULL)
2549: {
2550:   unsigned int flags = extra_data->flags;
2551:   if ((flags & PCRE_EXTRA_STUDY_DATA) != 0)
2552:     study = (const pcre_study_data *)extra_data->study_data;
2553:   if ((flags & PCRE_EXTRA_MATCH_LIMIT) != 0) return PCRE_ERROR_DFA_UMLIMIT;
2554:   if ((flags & PCRE_EXTRA_MATCH_LIMIT_RECURSION) != 0)
2555:     return PCRE_ERROR_DFA_UMLIMIT;
2556:   if ((flags & PCRE_EXTRA_CALLOUT_DATA) != 0)
2557:     md->callout_data = extra_data->callout_data;
2558:   if ((flags & PCRE_EXTRA_TABLES) != 0)

```

```

2559: md->tables = extra_data->tables;
2560: }
2561:
2562: /* Check that the first field in the block is the magic number. If it is not,
2563: test for a regex that was compiled on a host of opposite endianness. If this is
2564: the case, flipped values are put in internal_re and internal_study if there was
2565: study data too. */
2566:
2567: if (re->magic_number != MAGIC_NUMBER)
2568: {
2569: re = _pcre_try_flipped(re, &internal_re, study, &internal_study);
2570: if (re == NULL) return PCRE_ERROR_BADMAGIC;
2571: if (study != NULL) study = &internal_study;
2572: }
2573:
2574: /* Set some local values */
2575:
2576: current_subject = (const unsigned char *)subject + start_offset;
2577: end_subject = (const unsigned char *)subject + length;
2578: req_byte_ptr = current_subject - 1;
2579:
2580: #ifdef SUPPORT_UTF8
2581: utf8 = (re->options & PCRE_UTF8) != 0;
2582: #else
2583: utf8 = FALSE;
2584: #endif
2585:
2586: anchored = (options & (PCRE_ANCHORED|PCRE_DFA_RESTART)) != 0 ||
2587: (re->options & PCRE_ANCHORED) != 0;
2588:
2589: /* The remaining fixed data for passing around. */
2590:
2591: md->start_code = (const uschar *)argument_re +
2592: re->name_table_offset + re->name_count * re->name_entry_size;
2593: md->start_subject = (const unsigned char *)subject;
2594: md->end_subject = end_subject;
2595: md->moptions = options;
2596: md->poptions = re->options;
2597:
2598: /* If the BSR option is not set at match time, copy what was set
2599: at compile time. */

```

```

2600:
2601: if ((md->moptions & (PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE)) == 0)
2602: {
2603:   if ((re->options & (PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE)) != 0)
2604:     md->moptions |= re->options & (PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE);
2605: #ifdef BSR_ANYCRLF
2606:   else md->moptions |= PCRE_BSR_ANYCRLF;
2607: #endif
2608: }
2609:
2610: /* Handle different types of newline. The three bits give eight cases. If
2611: nothing is set at run time, whatever was used at compile time applies. */
2612:
2613: switch (((options & PCRE_NEWLINE_BITS) == 0)? re->options : (pcre_uint32)options) &
2614:     PCRE_NEWLINE_BITS)
2615: {
2616:   case 0: newline = NEWLINE; break; /* Compile-time default */
2617:   case PCRE_NEWLINE_CR: newline = '\r'; break;
2618:   case PCRE_NEWLINE_LF: newline = '\n'; break;
2619:   case PCRE_NEWLINE_CR+
2620:     PCRE_NEWLINE_LF: newline = ('\r' << 8) | '\n'; break;
2621:   case PCRE_NEWLINE_ANY: newline = -1; break;
2622:   case PCRE_NEWLINE_ANYCRLF: newline = -2; break;
2623:   default: return PCRE_ERROR_BADNEWLINE;
2624: }
2625:
2626: if (newline == -2)
2627: {
2628:   md->nltype = NLTYPE_ANYCRLF;
2629: }
2630: else if (newline < 0)
2631: {
2632:   md->nltype = NLTYPE_ANY;
2633: }
2634: else
2635: {
2636:   md->nltype = NLTYPE_FIXED;
2637:   if (newline > 255)
2638:   {
2639:     md->nllen = 2;
2640:     md->nl[0] = (newline >> 8) & 255;

```

```

2641:  md->nl[1] = newline & 255;
2642:  }
2643:  else
2644:  {
2645:    md->nllen = 1;
2646:    md->nl[0] = newline;
2647:  }
2648:  }
2649:
2650: /* Check a UTF-8 string if required. Unfortunately there's no way of passing
2651: back the character offset. */
2652:
2653: #ifdef SUPPORT_UTF8
2654: if (utf8 && (options & PCRE_NO_UTF8_CHECK) == 0)
2655: {
2656:   if (_pcre_valid_utf8((uschar *)subject, length) >= 0)
2657:     return PCRE_ERROR_BADUTF8;
2658:   if (start_offset > 0 && start_offset < length)
2659:   {
2660:     int tb = ((uschar *)subject)[start_offset];
2661:     if (tb > 127)
2662:     {
2663:       tb &= 0xc0;
2664:       if (tb != 0 && tb != 0xc0) return PCRE_ERROR_BADUTF8_OFFSET;
2665:     }
2666:   }
2667: }
2668: #endif
2669:
2670: /* If the exec call supplied NULL for tables, use the inbuilt ones. This
2671: is a feature that makes it possible to save compiled regex and re-use them
2672: in other programs later. */
2673:
2674: if (md->tables == NULL) md->tables = _pcre_default_tables;
2675:
2676: /* The lower casing table and the "must be at the start of a line" flag are
2677: used in a loop when finding where to start. */
2678:
2679: lcc = md->tables + lcc_offset;
2680: startline = (re->flags & PCRE_STARTLINE) != 0;
2681: firstline = (re->options & PCRE_FIRSTLINE) != 0;

```

```

2682:
2683: /* Set up the first character to match, if available. The first_byte value is
2684: never set for an anchored regular expression, but the anchoring may be forced
2685: at run time, so we have to test for anchoring. The first char may be unset for
2686: an unanchored pattern, of course. If there's no first char and the pattern was
2687: studied, there may be a bitmap of possible first characters. */
2688:
2689: if (!anchored)
2690: {
2691:   if ((re->flags & PCRE_FIRSTSET) != 0)
2692:   {
2693:     first_byte = re->first_byte & 255;
2694:     if ((first_byte_caseless = ((re->first_byte & REQ_CASELESS) != 0)) == TRUE)
2695:       first_byte = lcc[first_byte];
2696:   }
2697:   else
2698:   {
2699:     if (startline && study != NULL &&
2700:         (study->options & PCRE_STUDY_MAPPED) != 0)
2701:       start_bits = study->start_bits;
2702:   }
2703: }
2704:
2705: /* For anchored or unanchored matches, there may be a "last known required
2706: character" set. */
2707:
2708: if ((re->flags & PCRE_REQCHSET) != 0)
2709: {
2710:   req_byte = re->req_byte & 255;
2711:   req_byte_caseless = (re->req_byte & REQ_CASELESS) != 0;
2712:   req_byte2 = (md->tables + fcc_offset)[req_byte]; /* case flipped */
2713: }
2714:
2715: /* Call the main matching function, looping for a non-anchored regex after a
2716: failed match. Unless restarting, optimize by moving to the first match
2717: character if possible, when not anchored. Then unless wanting a partial match,
2718: check for a required later character. */
2719:
2720: for (;;)
2721: {
2722:   int rc;

```



```

2723:
2724: if ((options & PCRE_DFA_RESTART) == 0)
2725: {
2726:   const uschar *save_end_subject = end_subject;
2727:
2728:   /* Advance to a unique first char if possible. If firstline is TRUE, the
2729:   start of the match is constrained to the first line of a multiline string.
2730:   Implement this by temporarily adjusting end_subject so that we stop
2731:   scanning at a newline. If the match fails at the newline, later code breaks
2732:   this loop. */
2733:
2734:   if (firstline)
2735:   {
2736:     USPTR t = current_subject;
2737: #ifdef SUPPORT_UTF8
2738:     if (utf8)
2739:     {
2740:       while (t < md->end_subject && !IS_NEWLINE(t))
2741:       {
2742:         t++;
2743:         while (t < end_subject && (*t & 0xc0) == 0x80) t++;
2744:       }
2745:     }
2746:   else
2747: #endif
2748:     while (t < md->end_subject && !IS_NEWLINE(t)) t++;
2749:     end_subject = t;
2750:   }
2751:
2752:   if (first_byte >= 0)
2753:   {
2754:     if (first_byte_caseless)
2755:       while (current_subject < end_subject &&
2756:         lcc[*current_subject] != first_byte)
2757:         current_subject++;
2758:     else
2759:       while (current_subject < end_subject && *current_subject != first_byte)
2760:         current_subject++;
2761:   }
2762:
2763:   /* Or to just after a linebreak for a multiline match if possible */

```

```

2764:
2765:     else if (startline)
2766:     {
2767:         if (current_subject > md->start_subject + start_offset)
2768:         {
2769: #ifdef SUPPORT_UTF8
2770:         if (utf8)
2771:         {
2772:             while (current_subject < end_subject && !WAS_NEWLINE(current_subject))
2773:             {
2774:                 current_subject++;
2775:                 while(current_subject < end_subject &&
2776:                     (*current_subject & 0xc0) == 0x80)
2777:                     current_subject++;
2778:             }
2779:         }
2780:         else
2781: #endif
2782:         while (current_subject < end_subject && !WAS_NEWLINE(current_subject))
2783:             current_subject++;
2784:
2785:         /* If we have just passed a CR and the newline option is ANY or
2786:         ANYCRLF, and we are now at a LF, advance the match position by one more
2787:         character. */
2788:
2789:         if (current_subject[-1] == '\r' &&
2790:             (md->nltypes == NLTYPE_ANY || md->nltypes == NLTYPE_ANYCRLF) &&
2791:             current_subject < end_subject &&
2792:             *current_subject == '\n')
2793:             current_subject++;
2794:         }
2795:     }
2796:
2797:     /* Or to a non-unique first char after study */
2798:
2799:     else if (start_bits != NULL)
2800:     {
2801:         while (current_subject < end_subject)
2802:         {
2803:             register unsigned int c = *current_subject;
2804:             if (((start_bits[c/8] & (1 << (c&7))) == 0) current_subject++;

```

```

2805:     else break;
2806: }
2807: }
2808:
2809: /* Restore fudged end_subject */
2810:
2811: end_subject = save_end_subject;
2812: }
2813:
2814: /* If req_byte is set, we know that that character must appear in the subject
2815: for the match to succeed. If the first character is set, req_byte must be
2816: later in the subject; otherwise the test starts at the match point. This
2817: optimization can save a huge amount of work in patterns with nested unlimited
2818: repeats that aren't going to match. Writing separate code for cased/caseless
2819: versions makes it go faster, as does using an autoincrement and backing off
2820: on a match.
2821:
2822: HOWEVER: when the subject string is very, very long, searching to its end can
2823: take a long time, and give bad performance on quite ordinary patterns. This
2824: showed up when somebody was matching /^C/ on a 32-megabyte string... so we
2825: don't do this when the string is sufficiently long.
2826:
2827: ALSO: this processing is disabled when partial matching is requested.
2828: */
2829:
2830: if (req_byte >= 0 &&
2831:     end_subject - current_subject < REQ_BYTE_MAX &&
2832:     (options & PCRE_PARTIAL) == 0)
2833: {
2834:     register const uschar *p = current_subject + ((first_byte >= 0)? 1 : 0);
2835:
2836:     /* We don't need to repeat the search if we haven't yet reached the
2837:     place we found it at last time. */
2838:
2839:     if (p > req_byte_ptr)
2840:     {
2841:         if (req_byte_caseless)
2842:         {
2843:             while (p < end_subject)
2844:             {
2845:                 register int pp = *p++;

```

```

2846:     if (pp == req_byte || pp == req_byte2) { p--; break; }
2847:     }
2848:     }
2849:     else
2850:     {
2851:         while (p < end_subject)
2852:         {
2853:             if (*p++ == req_byte) { p--; break; }
2854:         }
2855:     }
2856:
2857:     /* If we can't find the required character, break the matching loop,
2858:     which will cause a return or PCRE_ERROR_NOMATCH. */
2859:
2860:     if (p >= end_subject) break;
2861:
2862:     /* If we have found the required character, save the point where we
2863:     found it, so that we don't search again next time round the loop if
2864:     the start hasn't passed this character yet. */
2865:
2866:     req_byte_ptr = p;
2867:     }
2868: }
2869:
2870: /* OK, now we can do the business */
2871:
2872: rc = internal_dfa_exec(
2873:     md,                /* fixed match data */
2874:     md->start_code,     /* this subexpression's code */
2875:     current_subject,    /* where we currently are */
2876:     start_offset,       /* start offset in subject */
2877:     offsets,            /* offset vector */
2878:     offsetcount,        /* size of same */
2879:     workspace,          /* workspace vector */
2880:     wscount,            /* size of same */
2881:     re->options & (PCRE_CASELESS|PCRE_MULTILINE|PCRE_DOTALL), /* ims flags */
2882:     0,                  /* function recurse level */
2883:     0);                 /* regex recurse level */
2884:
2885: /* Anything other than "no match" means we are done, always; otherwise, carry
2886: on only if not anchored. */

```

```

2887:
2888: if (rc != PCRE_ERROR_NOMATCH || anchored) return rc;
2889:
2890: /* Advance to the next subject character unless we are at the end of a line
2891: and firstline is set. */
2892:
2893: if (firstline && IS_NEWLINE(current_subject)) break;
2894: current_subject++;
2895: if (utf8)
2896: {
2897:   while (current_subject < end_subject && (*current_subject & 0xc0) == 0x80)
2898:     current_subject++;
2899: }
2900: if (current_subject > end_subject) break;
2901:
2902: /* If we have just passed a CR and we are now at a LF, and the pattern does
2903: not contain any explicit matches for \r or \n, and the newline option is CRLF
2904: or ANY or ANYCRLF, advance the match position by one more character. */
2905:
2906: if (current_subject[-1] == '\r' &&
2907:     current_subject < end_subject &&
2908:     *current_subject == '\n' &&
2909:     (re->flags & PCRE_HASCORLRF) == 0 &&
2910:     (md->nltypes == NLTYPE_ANY ||
2911:      md->nltypes == NLTYPE_ANYCRLF ||
2912:      md->nllen == 2))
2913:   current_subject++;
2914:
2915: } /* "Bumpalong" loop */
2916:
2917: return PCRE_ERROR_NOMATCH;
2918: }
2919:
2920: /* End of pcre_dfa_exec.c */

```

File: sdm/VxWorks/libRegex/pcre_try_flipped.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains an internal function that tests a compiled pattern to
42: see if it was compiled with the opposite endianness. If so, it uses an
43: auxiliary local function to flip the appropriate bytes. */
44:
45:
46: #ifdef HAVE_CONFIG_H
47: #include "config.h"
48: #endif
49:
50: #include "pcre_internal.h"
51:
52:
53: /*****
54:  *      Flip bytes in an integer      *
55:  *****/
56:
57: /* This function is called when the magic number in a regex doesn't match, in
58: order to flip its bytes to see if we are dealing with a pattern that was
59: compiled on a host of different endianness. If so, this function is used to
60: flip other byte values.
61:
62: Arguments:
63: value      the number to flip
64: n          the number of bytes to flip (assumed to be 2 or 4)
65:
66: Returns:   the flipped value
67: */
68:
69: static unsigned long int
70: byteflip(unsigned long int value, int n)
71: {
72: if (n == 2) return ((value & 0x00ff) << 8) | ((value & 0xff00) >> 8);
73: return ((value & 0x000000ff) << 24) |
74:        ((value & 0x0000ff00) << 8) |
75:        ((value & 0x00ff0000) >> 8) |
76:        ((value & 0xff000000) >> 24);

```

```

77: }
78:
79:
80:
81: /*****
82:  *    Test for a byte-flipped compiled regex  *
83: *****/
84:
85: /* This function is called from pcre_exec(), pcre_dfa_exec(), and also from
86: pcre_fullinfo(). Its job is to test whether the regex is byte-flipped - that
87: is, it was compiled on a system of opposite endianness. The function is called
88: only when the native MAGIC_NUMBER test fails. If the regex is indeed flipped,
89: we flip all the relevant values into a different data block, and return it.
90:
91: Arguments:
92: re          points to the regex
93: study       points to study data, or NULL
94: internal_re  points to a new regex block
95: internal_study points to a new study block
96:
97: Returns:     the new block if it is indeed a byte-flipped regex
98:             NULL if it is not
99: */
100:
101: real_pcre *
102: _pcre_try_flipped(const real_pcre *re, real_pcre *internal_re,
103: const pcre_study_data *study, pcre_study_data *internal_study)
104: {
105: if (byteflip(re->magic_number, sizeof(re->magic_number)) != MAGIC_NUMBER)
106: return NULL;
107:
108: *internal_re = *re;      /* To copy other fields */
109: internal_re->size = byteflip(re->size, sizeof(re->size));
110: internal_re->options = byteflip(re->options, sizeof(re->options));
111: internal_re->flags = (pcre_uint16)byteflip(re->flags, sizeof(re->flags));
112: internal_re->top_bracket =
113: (pcre_uint16)byteflip(re->top_bracket, sizeof(re->top_bracket));
114: internal_re->top_backref =
115: (pcre_uint16)byteflip(re->top_backref, sizeof(re->top_backref));
116: internal_re->first_byte =
117: (pcre_uint16)byteflip(re->first_byte, sizeof(re->first_byte));

```



```
118: internal_re->req_byte =
119: (pcr_uint16)byteflip(re->req_byte, sizeof(re->req_byte));
120: internal_re->name_table_offset =
121: (pcr_uint16)byteflip(re->name_table_offset, sizeof(re->name_table_offset));
122: internal_re->name_entry_size =
123: (pcr_uint16)byteflip(re->name_entry_size, sizeof(re->name_entry_size));
124: internal_re->name_count =
125: (pcr_uint16)byteflip(re->name_count, sizeof(re->name_count));
126:
127: if (study != NULL)
128: {
129: *internal_study = *study; /* To copy other fields */
130: internal_study->size = byteflip(study->size, sizeof(study->size));
131: internal_study->options = byteflip(study->options, sizeof(study->options));
132: }
133:
134: return internal_re;
135: }
136:
137: /* End of pcre_tryflipped.c */
```

File: sdm/VxWorks/libRegex/Makefile

```
1: # Wind River Workbench generated Makefile.
2: # Do not edit!!!
3: #
4: # The file ".wrmakefile" is the template used by the Wind River Workbench to
5: # generate the makefiles of this project. Add user-specific build targets and
6: # make rules only(!) in this project's ".wrmakefile" file. These will then be
7: # automatically dumped into the makefiles.
8:
9: WIND_HOME := $(subst \,,$(WIND_HOME))
10: WIND_BASE := $(subst \,,$(WIND_BASE))
11: WIND_USR := $(subst \,,$(WIND_USR))
12:
13: all : pre_recursion subdirs_all post_recursion pre_build main_all post_build
14:
15: TRACE=0
16: TRACEON=$(TRACE:0=@)
17: TRACE_FLAG=$(TRACEON:1=)
18:
19: MAKEFILE := Makefile
20:
21: BUILD_SPEC = PPC32gnu_RTP
22: DEBUG_MODE = 1
23: SRC_DIR := .
24: BUILD_ROOT_DIR :=
25: PRJ_ROOT_DIR := C:/Temp/Test/VxWorks/libRegex
26: WS_ROOT_DIR := C:/WindRiver/workspace
27:
28: ALL_BUILD_SPECS := ARMARCH5diab_RTP ARMARCH5diabbe_RTP ARMARCH5gnu_RTP
ARMARCH5gnube_RTP \
29: ARMARCH6diab_RTP ARMARCH6diabbe_RTP ARMARCH6gnu_RTP ARMARCH6gnube_RTP
\
30: MIPS32sfdiab_RTP MIPS32sfdiabbe_RTP MIPS32sfgnu_RTP MIPS32sfgnube_RTP \
31: MIPS64diab_RTP MIPS64diabbe_RTP MIPS64gnu_RTP MIPS64gnube_RTP \
32: PENTIUM2diab_RTP PENTIUM2gnu_RTP PENTIUM3diab_RTP PENTIUM3gnu_RTP \
33: PENTIUM4diab_RTP PENTIUM4gnu_RTP PENTIUMdiab_RTP PENTIUMgnu_RTP \
34: PPC32diab_RTP PPC32gnu_RTP PPC32sfdiab_RTP PPC32sfgnu_RTP \
35: SH32diab_RTP SH32diabbe_RTP SH32gnu_RTP SH32gnube_RTP \
36: SIMPENTIUMdiab_RTP SIMPENTIUMgnu_RTP SIMSPARCSOLARISdiab_RTP
SIMPARCSOLARISgnu_RTP \
37: XSCALEdiab_RTP XSCALEdiabbe_RTP XSCALEgnu_RTP XSCALEgnube_RTP
```

```

38: ENABLED_BUILD_SPECS := $(ALL_BUILD_SPECS)
39:
40: ifeq ($(BUILD_SPEC),ARMARCH5diab_RTP)
41: ifeq ($(DEBUG_MODE),1)
42: OBJ_DIR := ARMARCH5diab_RTP_DEBUG
43: else
44: OBJ_DIR := ARMARCH5diab_RTP
45: endif
46: endif
47: ifeq ($(BUILD_SPEC),ARMARCH5diabbe_RTP)
48: ifeq ($(DEBUG_MODE),1)
49: OBJ_DIR := ARMARCH5diabbe_RTP_DEBUG
50: else
51: OBJ_DIR := ARMARCH5diabbe_RTP
52: endif
53: endif
54: ifeq ($(BUILD_SPEC),ARMARCH5gnu_RTP)
55: ifeq ($(DEBUG_MODE),1)
56: OBJ_DIR := ARMARCH5gnu_RTP_DEBUG
57: else
58: OBJ_DIR := ARMARCH5gnu_RTP
59: endif
60: endif
61: ifeq ($(BUILD_SPEC),ARMARCH5gnube_RTP)
62: ifeq ($(DEBUG_MODE),1)
63: OBJ_DIR := ARMARCH5gnube_RTP_DEBUG
64: else
65: OBJ_DIR := ARMARCH5gnube_RTP
66: endif
67: endif
68: ifeq ($(BUILD_SPEC),ARMARCH6diab_RTP)
69: ifeq ($(DEBUG_MODE),1)
70: OBJ_DIR := ARMARCH6diab_RTP_DEBUG
71: else
72: OBJ_DIR := ARMARCH6diab_RTP
73: endif
74: endif
75: ifeq ($(BUILD_SPEC),ARMARCH6diabbe_RTP)
76: ifeq ($(DEBUG_MODE),1)
77: OBJ_DIR := ARMARCH6diabbe_RTP_DEBUG
78: else

```

```

79: OBJ_DIR := ARMARCH6diabbe_RTP
80: endif
81: endif
82: ifeq ($(BUILD_SPEC),ARMARCH6gnu_RTP)
83: ifeq ($(DEBUG_MODE),1)
84: OBJ_DIR := ARMARCH6gnu_RTP_DEBUG
85: else
86: OBJ_DIR := ARMARCH6gnu_RTP
87: endif
88: endif
89: ifeq ($(BUILD_SPEC),ARMARCH6gnube_RTP)
90: ifeq ($(DEBUG_MODE),1)
91: OBJ_DIR := ARMARCH6gnube_RTP_DEBUG
92: else
93: OBJ_DIR := ARMARCH6gnube_RTP
94: endif
95: endif
96: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
97: ifeq ($(DEBUG_MODE),1)
98: OBJ_DIR := MIPS32sfdiab_RTP_DEBUG
99: else
100: OBJ_DIR := MIPS32sfdiab_RTP
101: endif
102: endif
103: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
104: ifeq ($(DEBUG_MODE),1)
105: OBJ_DIR := MIPS32sfdiab_RTP_DEBUG
106: else
107: OBJ_DIR := MIPS32sfdiab_RTP
108: endif
109: endif
110: ifeq ($(BUILD_SPEC),MIPS32sfgnu_RTP)
111: ifeq ($(DEBUG_MODE),1)
112: OBJ_DIR := MIPS32sfgnu_RTP_DEBUG
113: else
114: OBJ_DIR := MIPS32sfgnu_RTP
115: endif
116: endif
117: ifeq ($(BUILD_SPEC),MIPS32sfgnule_RTP)
118: ifeq ($(DEBUG_MODE),1)
119: OBJ_DIR := MIPS32sfgnule_RTP_DEBUG

```

```

120: else
121: OBJ_DIR := MIPS32sfgnule_RTP
122: endif
123: endif
124: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
125: ifeq ($(DEBUG_MODE),1)
126: OBJ_DIR := MIPS64diab_RTP_DEBUG
127: else
128: OBJ_DIR := MIPS64diab_RTP
129: endif
130: endif
131: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
132: ifeq ($(DEBUG_MODE),1)
133: OBJ_DIR := MIPS64diab_RTP_DEBUG
134: else
135: OBJ_DIR := MIPS64diab_RTP
136: endif
137: endif
138: ifeq ($(BUILD_SPEC),MIPS64gnu_RTP)
139: ifeq ($(DEBUG_MODE),1)
140: OBJ_DIR := MIPS64gnu_RTP_DEBUG
141: else
142: OBJ_DIR := MIPS64gnu_RTP
143: endif
144: endif
145: ifeq ($(BUILD_SPEC),MIPS64gnule_RTP)
146: ifeq ($(DEBUG_MODE),1)
147: OBJ_DIR := MIPS64gnule_RTP_DEBUG
148: else
149: OBJ_DIR := MIPS64gnule_RTP
150: endif
151: endif
152: ifeq ($(BUILD_SPEC),PENTIUM2diab_RTP)
153: ifeq ($(DEBUG_MODE),1)
154: OBJ_DIR := PENTIUM2diab_RTP_DEBUG
155: else
156: OBJ_DIR := PENTIUM2diab_RTP
157: endif
158: endif
159: ifeq ($(BUILD_SPEC),PENTIUM2gnu_RTP)
160: ifeq ($(DEBUG_MODE),1)

```

```

161: OBJ_DIR := PENTIUM2gnu_RTP_DEBUG
162: else
163: OBJ_DIR := PENTIUM2gnu_RTP
164: endif
165: endif
166: ifeq ($(BUILD_SPEC),PENTIUM3diab_RTP)
167: ifeq ($(DEBUG_MODE),1)
168: OBJ_DIR := PENTIUM3diab_RTP_DEBUG
169: else
170: OBJ_DIR := PENTIUM3diab_RTP
171: endif
172: endif
173: ifeq ($(BUILD_SPEC),PENTIUM3gnu_RTP)
174: ifeq ($(DEBUG_MODE),1)
175: OBJ_DIR := PENTIUM3gnu_RTP_DEBUG
176: else
177: OBJ_DIR := PENTIUM3gnu_RTP
178: endif
179: endif
180: ifeq ($(BUILD_SPEC),PENTIUM4diab_RTP)
181: ifeq ($(DEBUG_MODE),1)
182: OBJ_DIR := PENTIUM4diab_RTP_DEBUG
183: else
184: OBJ_DIR := PENTIUM4diab_RTP
185: endif
186: endif
187: ifeq ($(BUILD_SPEC),PENTIUM4gnu_RTP)
188: ifeq ($(DEBUG_MODE),1)
189: OBJ_DIR := PENTIUM4gnu_RTP_DEBUG
190: else
191: OBJ_DIR := PENTIUM4gnu_RTP
192: endif
193: endif
194: ifeq ($(BUILD_SPEC),PENTIUMdiab_RTP)
195: ifeq ($(DEBUG_MODE),1)
196: OBJ_DIR := PENTIUMdiab_RTP_DEBUG
197: else
198: OBJ_DIR := PENTIUMdiab_RTP
199: endif
200: endif
201: ifeq ($(BUILD_SPEC),PENTIUMgnu_RTP)

```

```

202: ifeq ($(DEBUG_MODE),1)
203: OBJ_DIR := PENTIUMgnu_RTP_DEBUG
204: else
205: OBJ_DIR := PENTIUMgnu_RTP
206: endif
207: endif
208: ifeq ($(BUILD_SPEC),PPC32diab_RTP)
209: ifeq ($(DEBUG_MODE),1)
210: OBJ_DIR := PPC32diab_RTP_DEBUG
211: else
212: OBJ_DIR := PPC32diab_RTP
213: endif
214: endif
215: ifeq ($(BUILD_SPEC),PPC32gnu_RTP)
216: ifeq ($(DEBUG_MODE),1)
217: OBJ_DIR := ../bin/regex_DEBUG
218: else
219: OBJ_DIR := ../bin/regex
220: endif
221: endif
222: ifeq ($(BUILD_SPEC),PPC32sfdiab_RTP)
223: ifeq ($(DEBUG_MODE),1)
224: OBJ_DIR := PPC32sfdiab_RTP_DEBUG
225: else
226: OBJ_DIR := PPC32sfdiab_RTP
227: endif
228: endif
229: ifeq ($(BUILD_SPEC),PPC32sfgnu_RTP)
230: ifeq ($(DEBUG_MODE),1)
231: OBJ_DIR := PPC32sfgnu_RTP_DEBUG
232: else
233: OBJ_DIR := PPC32sfgnu_RTP
234: endif
235: endif
236: ifeq ($(BUILD_SPEC),SH32diab_RTP)
237: ifeq ($(DEBUG_MODE),1)
238: OBJ_DIR := SH32diab_RTP_DEBUG
239: else
240: OBJ_DIR := SH32diab_RTP
241: endif
242: endif

```

```

243: ifeq ($(BUILD_SPEC),SH32diab_RTP)
244: ifeq ($(DEBUG_MODE),1)
245: OBJ_DIR := SH32diab_RTP_DEBUG
246: else
247: OBJ_DIR := SH32diab_RTP
248: endif
249: endif
250: ifeq ($(BUILD_SPEC),SH32gnu_RTP)
251: ifeq ($(DEBUG_MODE),1)
252: OBJ_DIR := SH32gnu_RTP_DEBUG
253: else
254: OBJ_DIR := SH32gnu_RTP
255: endif
256: endif
257: ifeq ($(BUILD_SPEC),SH32gnule_RTP)
258: ifeq ($(DEBUG_MODE),1)
259: OBJ_DIR := SH32gnule_RTP_DEBUG
260: else
261: OBJ_DIR := SH32gnule_RTP
262: endif
263: endif
264: ifeq ($(BUILD_SPEC),SIMPENTIUMdiab_RTP)
265: ifeq ($(DEBUG_MODE),1)
266: OBJ_DIR := SIMPENTIUMdiab_RTP_DEBUG
267: else
268: OBJ_DIR := SIMPENTIUMdiab_RTP
269: endif
270: endif
271: ifeq ($(BUILD_SPEC),SIMPENTIUMgnu_RTP)
272: ifeq ($(DEBUG_MODE),1)
273: OBJ_DIR := SIMPENTIUMgnu_RTP_DEBUG
274: else
275: OBJ_DIR := SIMPENTIUMgnu_RTP
276: endif
277: endif
278: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISdiab_RTP)
279: ifeq ($(DEBUG_MODE),1)
280: OBJ_DIR := SIMSPARCSOLARISdiab_RTP_DEBUG
281: else
282: OBJ_DIR := SIMSPARCSOLARISdiab_RTP
283: endif

```



```

284: endif
285: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISgnu_RTP)
286: ifeq ($(DEBUG_MODE),1)
287: OBJ_DIR := SIMSPARCSOLARISgnu_RTP_DEBUG
288: else
289: OBJ_DIR := SIMSPARCSOLARISgnu_RTP
290: endif
291: endif
292: ifeq ($(BUILD_SPEC),XSCALEdiab_RTP)
293: ifeq ($(DEBUG_MODE),1)
294: OBJ_DIR := XSCALEdiab_RTP_DEBUG
295: else
296: OBJ_DIR := XSCALEdiab_RTP
297: endif
298: endif
299: ifeq ($(BUILD_SPEC),XSCALEdiabbe_RTP)
300: ifeq ($(DEBUG_MODE),1)
301: OBJ_DIR := XSCALEdiabbe_RTP_DEBUG
302: else
303: OBJ_DIR := XSCALEdiabbe_RTP
304: endif
305: endif
306: ifeq ($(BUILD_SPEC),XSCALEgnu_RTP)
307: ifeq ($(DEBUG_MODE),1)
308: OBJ_DIR := XSCALEgnu_RTP_DEBUG
309: else
310: OBJ_DIR := XSCALEgnu_RTP
311: endif
312: endif
313: ifeq ($(BUILD_SPEC),XSCALEgnube_RTP)
314: ifeq ($(DEBUG_MODE),1)
315: OBJ_DIR := XSCALEgnube_RTP_DEBUG
316: else
317: OBJ_DIR := XSCALEgnube_RTP
318: endif
319: endif
320:
321: DEP_FILES := $(OBJ_DIR)/pcre_get.d $(OBJ_DIR)/pcre_ucd.d $(OBJ_DIR)/pcre_ord2utf8.d \
322:   $(OBJ_DIR)/pcre_newline.d $(OBJ_DIR)/pcre_xclass.d $(OBJ_DIR)/pcre_chartables.d \
323:   $(OBJ_DIR)/pcre_globals.d $(OBJ_DIR)/pcre_compile.d $(OBJ_DIR)/pcre_exec.d \
324:   $(OBJ_DIR)/pcre_info.d $(OBJ_DIR)/pcrecpp.d $(OBJ_DIR)/pcre_config.d \

```

```

325: $(OBJ_DIR)/pcre_dfa_exec.d $(OBJ_DIR)/pcre_refcount.d $(OBJ_DIR)/pcreposix.d \
326: $(OBJ_DIR)/pcre_tables.d $(OBJ_DIR)/pcre_maketables.d $(OBJ_DIR)/pcre_version.d \
327: $(OBJ_DIR)/pcre_try_flipped.d $(OBJ_DIR)/pcre_study.d $(OBJ_DIR)/pcre_fullinfo.d \
328: $(OBJ_DIR)/pcre_valid_utf8.d
329: -include $(DEP_FILES)
330:
331: ifeq ($(BUILD_SPEC),ARMARCH5diab_RTP)
332: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
333: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
334: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
335: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
336: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
337: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
338: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
339: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
340: $(OBJ_DIR)/pcre_valid_utf8.sho
341:
342: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
343:
344: SUB_OBJECTS :=
345: SUB_TARGETS :=
346: endif
347: ifeq ($(BUILD_SPEC),ARMARCH5diabbe_RTP)
348: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
349: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
350: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
351: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
352: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
353: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
354: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
355: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
356: $(OBJ_DIR)/pcre_valid_utf8.sho
357:
358: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
359:
360: SUB_OBJECTS :=
361: SUB_TARGETS :=
362: endif
363: ifeq ($(BUILD_SPEC),ARMARCH5gnu_RTP)

```

```

364: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
365: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
366:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
367:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
368:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
369:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
370:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
371:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
372:    $(OBJ_DIR)/pcre_valid_utf8.sho
373:
374: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
375:
376: SUB_OBJECTS :=
377: SUB_TARGETS :=
378: endif
379: ifeq ($(BUILD_SPEC),ARMARCH5gnube_RTP)
380: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
381: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
382:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
383:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
384:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
385:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
386:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
387:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
388:    $(OBJ_DIR)/pcre_valid_utf8.sho
389:
390: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
391:
392: SUB_OBJECTS :=
393: SUB_TARGETS :=
394: endif
395: ifeq ($(BUILD_SPEC),ARMARCH6diab_RTP)
396: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
397: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
398:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
399:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
400:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
401:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
402:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \

```

```

403: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
404: $(OBJ_DIR)/pcre_valid_utf8.sho
405:
406: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
407:
408: SUB_OBJECTS :=
409: SUB_TARGETS :=
410: endif
411: ifeq ($(BUILD_SPEC),ARMARCH6diabbe_RTP)
412: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
413: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
414: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
415: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
416: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
417: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
418: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
419: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
420: $(OBJ_DIR)/pcre_valid_utf8.sho
421:
422: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
423:
424: SUB_OBJECTS :=
425: SUB_TARGETS :=
426: endif
427: ifeq ($(BUILD_SPEC),ARMARCH6gnu_RTP)
428: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
429: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
430: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
431: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
432: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
433: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
434: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
435: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
436: $(OBJ_DIR)/pcre_valid_utf8.sho
437:
438: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
439:
440: SUB_OBJECTS :=
441: SUB_TARGETS :=

```

```

442: endif
443: ifeq ($(BUILD_SPEC),ARMARCH6gnube_RTP)
444: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
445: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
446:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
447:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
448:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
449:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
450:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
451:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
452:    $(OBJ_DIR)/pcre_valid_utf8.sho
453:
454: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
455:
456: SUB_OBJECTS :=
457: SUB_TARGETS :=
458: endif
459: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
460: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
461: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
462:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
463:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
464:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
465:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
466:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
467:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
468:    $(OBJ_DIR)/pcre_valid_utf8.sho
469:
470: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
471:
472: SUB_OBJECTS :=
473: SUB_TARGETS :=
474: endif
475: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
476: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
477: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
478:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
479:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
480:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \

```

```

481: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
482: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
483: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
484: $(OBJ_DIR)/pcre_valid_utf8.sho
485:
486: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
487:
488: SUB_OBJECTS :=
489: SUB_TARGETS :=
490: endif
491: ifeq ($(BUILD_SPEC),MIPS32sfgnu_RTP)
492: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
493: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
494: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
495: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
496: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
497: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
498: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
499: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
500: $(OBJ_DIR)/pcre_valid_utf8.sho
501:
502: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
503:
504: SUB_OBJECTS :=
505: SUB_TARGETS :=
506: endif
507: ifeq ($(BUILD_SPEC),MIPS32sfgnule_RTP)
508: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
509: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
510: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
511: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
512: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
513: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
514: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
515: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
516: $(OBJ_DIR)/pcre_valid_utf8.sho
517:
518: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
519:

```

```

520: SUB_OBJECTS :=
521: SUB_TARGETS :=
522: endif
523: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
524: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
525: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucl.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
526:     $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
527:     $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
528:     $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
529:     $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
530:     $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
531:     $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
532:     $(OBJ_DIR)/pcre_valid_utf8.sho
533:
534: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
535:
536: SUB_OBJECTS :=
537: SUB_TARGETS :=
538: endif
539: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
540: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
541: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucl.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
542:     $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
543:     $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
544:     $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
545:     $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
546:     $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
547:     $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
548:     $(OBJ_DIR)/pcre_valid_utf8.sho
549:
550: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
551:
552: SUB_OBJECTS :=
553: SUB_TARGETS :=
554: endif
555: ifeq ($(BUILD_SPEC),MIPS64gnu_RTP)
556: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
557: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucl.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
558:     $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \

```

```

559: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
560: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
561: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
562: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
563: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
564: $(OBJ_DIR)/pcre_valid_utf8.sho
565:
566: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
567:
568: SUB_OBJECTS :=
569: SUB_TARGETS :=
570: endif
571: ifeq ($(BUILD_SPEC),MIPS64gnule_RTP)
572: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
573: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
574: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
575: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
576: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
577: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
578: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
579: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
580: $(OBJ_DIR)/pcre_valid_utf8.sho
581:
582: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
583:
584: SUB_OBJECTS :=
585: SUB_TARGETS :=
586: endif
587: ifeq ($(BUILD_SPEC),PENTIUM2diab_RTP)
588: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
589: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
590: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
591: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
592: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
593: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
594: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
595: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
596: $(OBJ_DIR)/pcre_valid_utf8.sho
597:

```



```

598: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
599:
600: SUB_OBJECTS :=
601: SUB_TARGETS :=
602: endif
603: ifeq ($(BUILD_SPEC),PENTIUM2gnu_RTP)
604: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
605: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
606: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
607: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
608: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
609: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
610: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
611: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
612: $(OBJ_DIR)/pcre_valid_utf8.sho
613:
614: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
615:
616: SUB_OBJECTS :=
617: SUB_TARGETS :=
618: endif
619: ifeq ($(BUILD_SPEC),PENTIUM3diab_RTP)
620: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
621: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
622: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
623: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
624: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
625: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
626: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
627: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
628: $(OBJ_DIR)/pcre_valid_utf8.sho
629:
630: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
631:
632: SUB_OBJECTS :=
633: SUB_TARGETS :=
634: endif
635: ifeq ($(BUILD_SPEC),PENTIUM3gnu_RTP)
636: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"

```

```

637: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho
\
638:   $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
639:   $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
640:   $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
641:   $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
642:   $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
643:   $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
644:   $(OBJ_DIR)/pcre_valid_utf8.sho
645:
646: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
647:
648: SUB_OBJECTS :=
649: SUB_TARGETS :=
650: endif
651: ifeq ($(BUILD_SPEC),PENTIUM4diab_RTP)
652: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
653: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho
\
654:   $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
655:   $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
656:   $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
657:   $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
658:   $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
659:   $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
660:   $(OBJ_DIR)/pcre_valid_utf8.sho
661:
662: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
663:
664: SUB_OBJECTS :=
665: SUB_TARGETS :=
666: endif
667: ifeq ($(BUILD_SPEC),PENTIUM4gnu_RTP)
668: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
669: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho
\
670:   $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
671:   $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
672:   $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
673:   $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
674:   $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
675:   $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \

```

```

676:    $(OBJ_DIR)/pcre_valid_utf8.sho
677:
678: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
679:
680: SUB_OBJECTS :=
681: SUB_TARGETS :=
682: endif
683: ifeq ($(BUILD_SPEC),PENTIUMdiab_RTP)
684: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
685: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
686:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
687:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
688:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
689:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
690:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
691:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
692:    $(OBJ_DIR)/pcre_valid_utf8.sho
693:
694: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
695:
696: SUB_OBJECTS :=
697: SUB_TARGETS :=
698: endif
699: ifeq ($(BUILD_SPEC),PENTIUMgnu_RTP)
700: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
701: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
702:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
703:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
704:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
705:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
706:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
707:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
708:    $(OBJ_DIR)/pcre_valid_utf8.sho
709:
710: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
711:
712: SUB_OBJECTS :=
713: SUB_TARGETS :=
714: endif

```

```

715: ifeq ($(BUILD_SPEC),PPC32diab_RTP)
716: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
717: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
718:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
719:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
720:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
721:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
722:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
723:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
724:    $(OBJ_DIR)/pcre_valid_utf8.sho
725:
726: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
727:
728: SUB_OBJECTS :=
729: SUB_TARGETS :=
730: endif
731: ifeq ($(BUILD_SPEC),PPC32gnu_RTP)
732: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
733: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
734:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
735:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
736:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
737:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
738:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
739:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
740:    $(OBJ_DIR)/pcre_valid_utf8.sho
741:
742: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
743:
744: SUB_OBJECTS :=
745: SUB_TARGETS :=
746: endif
747: ifeq ($(BUILD_SPEC),PPC32sfdiab_RTP)
748: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
749: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
750:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
751:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
752:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
753:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \

```

```

754: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
755: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
756: $(OBJ_DIR)/pcre_valid_utf8.sho
757:
758: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
759:
760: SUB_OBJECTS :=
761: SUB_TARGETS :=
762: endif
763: ifeq ($(BUILD_SPEC),PPC32sfgnu_RTP)
764: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
765: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
766: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
767: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
768: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
769: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
770: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
771: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
772: $(OBJ_DIR)/pcre_valid_utf8.sho
773:
774: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
775:
776: SUB_OBJECTS :=
777: SUB_TARGETS :=
778: endif
779: ifeq ($(BUILD_SPEC),SH32diab_RTP)
780: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
781: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
782: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
783: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
784: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
785: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
786: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
787: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
788: $(OBJ_DIR)/pcre_valid_utf8.sho
789:
790: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
791:
792: SUB_OBJECTS :=

```

```

793: SUB_TARGETS :=
794: endif
795: ifeq ($(BUILD_SPEC),SH32diablib_RTP)
796: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
797: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
798:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
799:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
800:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
801:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
802:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
803:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
804:    $(OBJ_DIR)/pcre_valid_utf8.sho
805:
806: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
807:
808: SUB_OBJECTS :=
809: SUB_TARGETS :=
810: endif
811: ifeq ($(BUILD_SPEC),SH32gnu_RTP)
812: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
813: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
814:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
815:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
816:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
817:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
818:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
819:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
820:    $(OBJ_DIR)/pcre_valid_utf8.sho
821:
822: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
823:
824: SUB_OBJECTS :=
825: SUB_TARGETS :=
826: endif
827: ifeq ($(BUILD_SPEC),SH32gnulib_RTP)
828: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
829: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
830:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
831:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \

```

```

832: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
833: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
834: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
835: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
836: $(OBJ_DIR)/pcre_valid_utf8.sho
837:
838: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
839:
840: SUB_OBJECTS :=
841: SUB_TARGETS :=
842: endif
843: ifeq ($(BUILD_SPEC),SIMPENTIUMdiab_RTP)
844: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
845: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
846: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
847: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
848: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
849: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
850: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
851: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
852: $(OBJ_DIR)/pcre_valid_utf8.sho
853:
854: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
855:
856: SUB_OBJECTS :=
857: SUB_TARGETS :=
858: endif
859: ifeq ($(BUILD_SPEC),SIMPENTIUMgnu_RTP)
860: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
861: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
862: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
863: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
864: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
865: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
866: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
867: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
868: $(OBJ_DIR)/pcre_valid_utf8.sho
869:
870: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a

```

```

871:
872: SUB_OBJECTS :=
873: SUB_TARGETS :=
874: endif
875: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISdiab_RTP)
876: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
877: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
878:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
879:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
880:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
881:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
882:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
883:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
884:    $(OBJ_DIR)/pcre_valid_utf8.sho
885:
886: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
887:
888: SUB_OBJECTS :=
889: SUB_TARGETS :=
890: endif
891: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISgnu_RTP)
892: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
893: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
894:    $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
895:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
896:    $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
897:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
898:    $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
899:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
900:    $(OBJ_DIR)/pcre_valid_utf8.sho
901:
902: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
903:
904: SUB_OBJECTS :=
905: SUB_TARGETS :=
906: endif
907: ifeq ($(BUILD_SPEC),XSCALEdiab_RTP)
908: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
909: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\

```



```

910: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
911: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
912: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
913: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
914: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
915: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
916: $(OBJ_DIR)/pcre_valid_utf8.sho
917:
918: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
919:
920: SUB_OBJECTS :=
921: SUB_TARGETS :=
922: endif
923: ifeq ($(BUILD_SPEC),XSCALEdiabbe_RTP)
924: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
925: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucl.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
926: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
927: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
928: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
929: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
930: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
931: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
932: $(OBJ_DIR)/pcre_valid_utf8.sho
933:
934: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
935:
936: SUB_OBJECTS :=
937: SUB_TARGETS :=
938: endif
939: ifeq ($(BUILD_SPEC),XSCALEgnu_RTP)
940: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
941: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucl.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
942: $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
943: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
944: $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
945: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
946: $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
947: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
948: $(OBJ_DIR)/pcre_valid_utf8.sho

```

```

949:
950: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
951:
952: SUB_OBJECTS :=
953: SUB_TARGETS :=
954: endif
955: ifeq ($(BUILD_SPEC),XSCALEgnube_RTP)
956: SUBDIRS := "C:/Temp/Test/VxWorks/libRegex/.svn"
957: OBJECTS := $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho $(OBJ_DIR)/pcre_ord2utf8.sho \
\
958:     $(OBJ_DIR)/pcre_newline.sho $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
959:     $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho $(OBJ_DIR)/pcre_exec.sho \
960:     $(OBJ_DIR)/pcre_info.sho $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
961:     $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho $(OBJ_DIR)/pcreposix.sho \
962:     $(OBJ_DIR)/pcre_tables.sho $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
963:     $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho $(OBJ_DIR)/pcre_fullinfo.sho \
964:     $(OBJ_DIR)/pcre_valid_utf8.sho
965:
966: PROJECT_TARGETS := $(OBJ_DIR)/libRegex.a
967:
968: SUB_OBJECTS :=
969: SUB_TARGETS :=
970: endif
971:
972: PROJECT_TYPE = SL
973: DEFINES = -DHAVE_CONFIG_H
974: DO_STRIP = 0
975: SHAREDLIB_VERSION =
976: EXPAND_DBG = 0
977:
978: ifeq ($(BUILD_SPEC),ARMARCH5diab_RTP)
979: VX_CPU_FAMILY = arm
980: CPU = ARMARCH5
981: TOOL_FAMILY = diab
982: TOOL = diab
983: TOOL_PATH =
984: CC_ARCH_SPEC = -tARMV5LS:rtplib
985: LIBPATH =
986: LIBS = -lstdc++
987:

```

```

988:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
989: endif
990:
991: ifeq ($(BUILD_SPEC),ARMARCH5diabbe_RTP)
992: VX_CPU_FAMILY = arm
993: CPU = ARMARCH5
994: TOOL_FAMILY = diab
995: TOOL = diabbe
996: TOOL_PATH =
997: CC_ARCH_SPEC = -tARMV5ES:rtp
998: LIBPATH =
999: LIBS = -lstdlstd
1000:
1001:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1002: endif
1003:
1004: ifeq ($(BUILD_SPEC),ARMARCH5gnu_RTP)
1005: VX_CPU_FAMILY = arm
1006: CPU = ARMARCH5
1007: TOOL_FAMILY = gnu
1008: TOOL = gnu
1009: TOOL_PATH =
1010: CC_ARCH_SPEC = -t5
1011: LIBPATH =
1012: LIBS = -lstdc++
1013:
1014:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1015: endif
1016:
1017: ifeq ($(BUILD_SPEC),ARMARCH5gnube_RTP)
1018: VX_CPU_FAMILY = arm
1019: CPU = ARMARCH5
1020: TOOL_FAMILY = gnu
1021: TOOL = gnube
1022: TOOL_PATH =
1023: CC_ARCH_SPEC = -t5be -Wa,-EB
1024: LIBPATH =
1025: LIBS = -lstdc++
1026:

```

```

1027: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1028: endif
1029:
1030: ifeq ($(BUILD_SPEC),ARMARCH6diab_RTP)
1031: VX_CPU_FAMILY = arm
1032: CPU = ARMARCH6
1033: TOOL_FAMILY = diab
1034: TOOL = diab
1035: TOOL_PATH =
1036: CC_ARCH_SPEC = -tARMV6LS:rtp
1037: LIBPATH =
1038: LIBS = -lstdlstd
1039:
1040: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1041: endif
1042:
1043: ifeq ($(BUILD_SPEC),ARMARCH6diabbe_RTP)
1044: VX_CPU_FAMILY = arm
1045: CPU = ARMARCH6
1046: TOOL_FAMILY = diab
1047: TOOL = diabbe
1048: TOOL_PATH =
1049: CC_ARCH_SPEC = -tARMV6ES:rtp
1050: LIBPATH =
1051: LIBS = -lstdlstd
1052:
1053: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1054: endif
1055:
1056: ifeq ($(BUILD_SPEC),ARMARCH6gnu_RTP)
1057: VX_CPU_FAMILY = arm
1058: CPU = ARMARCH6
1059: TOOL_FAMILY = gnu
1060: TOOL = gnu
1061: TOOL_PATH =
1062: CC_ARCH_SPEC = -t5
1063: LIBPATH =
1064: LIBS = -lstdc++
1065:

```

```

1066: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1067: endif
1068:
1069: ifeq ($(BUILD_SPEC),ARMARCH6gnube_RTP)
1070: VX_CPU_FAMILY = arm
1071: CPU = ARMARCH6
1072: TOOL_FAMILY = gnu
1073: TOOL = gnube
1074: TOOL_PATH =
1075: CC_ARCH_SPEC = -t5be -Wa,-EB
1076: LIBPATH =
1077: LIBS = -lstdc++
1078:
1079: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1080: endif
1081:
1082: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
1083: VX_CPU_FAMILY = mips
1084: CPU = MIPS32
1085: TOOL_FAMILY = diab
1086: TOOL = sfdiab
1087: TOOL_PATH =
1088: CC_ARCH_SPEC = -tMIPS2FS:rtp
1089: LIBPATH =
1090: LIBS = -lstdc++
1091:
1092: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1093: endif
1094:
1095: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
1096: VX_CPU_FAMILY = mips
1097: CPU = MIPS32
1098: TOOL_FAMILY = diab
1099: TOOL = sfdiab
1100: TOOL_PATH =
1101: CC_ARCH_SPEC = -tMIPS2MS:rtp
1102: LIBPATH =
1103: LIBS = -lstdc++
1104:

```

```

1105:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1106: endif
1107:
1108: ifeq ($(BUILD_SPEC),MIPS32sfgnu_RTP)
1109: VX_CPU_FAMILY = mips
1110: CPU = MIPS32
1111: TOOL_FAMILY = gnu
1112: TOOL = sfgnu
1113: TOOL_PATH =
1114: CC_ARCH_SPEC = -G 0 -mno-branch-likely -mips2 -EB
1115: LIBPATH =
1116: LIBS = -lstdc++
1117:
1118:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1119: endif
1120:
1121: ifeq ($(BUILD_SPEC),MIPS32sfgnule_RTP)
1122: VX_CPU_FAMILY = mips
1123: CPU = MIPS32
1124: TOOL_FAMILY = gnu
1125: TOOL = sfgnule
1126: TOOL_PATH =
1127: CC_ARCH_SPEC = -G 0 -mno-branch-likely -mips2 -EL
1128: LIBPATH =
1129: LIBS = -lstdc++
1130:
1131:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1132: endif
1133:
1134: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
1135: VX_CPU_FAMILY = mips
1136: CPU = MIPS64
1137: TOOL_FAMILY = diab
1138: TOOL = diab
1139: TOOL_PATH =
1140: CC_ARCH_SPEC = -tMIPS3XH:rtp
1141: LIBPATH =
1142: LIBS = -lstd
1143:

```

```

1144:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1145: endif
1146:
1147: ifeq ($(BUILD_SPEC),MIPS64diab RTP)
1148: VX_CPU_FAMILY = mips
1149: CPU = MIPS64
1150: TOOL_FAMILY = diab
1151: TOOL = diab
1152: TOOL_PATH =
1153: CC_ARCH_SPEC = -tMIPS3ZH:rtp
1154: LIBPATH =
1155: LIBS = -lstd
1156:
1157:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1158: endif
1159:
1160: ifeq ($(BUILD_SPEC),MIPS64gnu RTP)
1161: VX_CPU_FAMILY = mips
1162: CPU = MIPS64
1163: TOOL_FAMILY = gnu
1164: TOOL = gnu
1165: TOOL_PATH =
1166: CC_ARCH_SPEC = -G 0 -mno-branch-likely -mips3 -mabi=o64 -mcp64 -EB
1167: LIBPATH =
1168: LIBS = -lstdc++
1169:
1170:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1171: endif
1172:
1173: ifeq ($(BUILD_SPEC),MIPS64gnule RTP)
1174: VX_CPU_FAMILY = mips
1175: CPU = MIPS64
1176: TOOL_FAMILY = gnu
1177: TOOL = gnule
1178: TOOL_PATH =
1179: CC_ARCH_SPEC = -G 0 -mno-branch-likely -mips3 -mabi=o64 -mcp64 -EL
1180: LIBPATH =
1181: LIBS = -lstdc++
1182:

```

```

1183:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1184: endif
1185:
1186: ifeq ($(BUILD_SPEC),PENTIUM2diab_RTP)
1187: VX_CPU_FAMILY = pentium
1188: CPU = PENTIUM2
1189: TOOL_FAMILY = diab
1190: TOOL = diab
1191: TOOL_PATH =
1192: CC_ARCH_SPEC = -tPENTIUM2LH:rtp
1193: LIBPATH =
1194: LIBS = -lstdlstd
1195:
1196:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1197: endif
1198:
1199: ifeq ($(BUILD_SPEC),PENTIUM2gnu_RTP)
1200: VX_CPU_FAMILY = pentium
1201: CPU = PENTIUM2
1202: TOOL_FAMILY = gnu
1203: TOOL = gnu
1204: TOOL_PATH =
1205: CC_ARCH_SPEC = -mtune=i486 -march=i486
1206: LIBPATH =
1207: LIBS = -lstdc++
1208:
1209:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1210: endif
1211:
1212: ifeq ($(BUILD_SPEC),PENTIUM3diab_RTP)
1213: VX_CPU_FAMILY = pentium
1214: CPU = PENTIUM3
1215: TOOL_FAMILY = diab
1216: TOOL = diab
1217: TOOL_PATH =
1218: CC_ARCH_SPEC = -tPENTIUM3LH:rtp
1219: LIBPATH =
1220: LIBS = -lstdlstd
1221:

```



```

1222:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1223: endif
1224:
1225: ifeq ($(BUILD_SPEC),PENTIUM3gnu_RTP)
1226: VX_CPU_FAMILY = pentium
1227: CPU = PENTIUM3
1228: TOOL_FAMILY = gnu
1229: TOOL = gnu
1230: TOOL_PATH =
1231: CC_ARCH_SPEC = -mtune=i486 -march=i486
1232: LIBPATH =
1233: LIBS = -lstdc++
1234:
1235:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1236: endif
1237:
1238: ifeq ($(BUILD_SPEC),PENTIUM4diab_RTP)
1239: VX_CPU_FAMILY = pentium
1240: CPU = PENTIUM4
1241: TOOL_FAMILY = diab
1242: TOOL = diab
1243: TOOL_PATH =
1244: CC_ARCH_SPEC = -tPENTIUM4LH:rtp
1245: LIBPATH =
1246: LIBS = -lstdc++
1247:
1248:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1249: endif
1250:
1251: ifeq ($(BUILD_SPEC),PENTIUM4gnu_RTP)
1252: VX_CPU_FAMILY = pentium
1253: CPU = PENTIUM4
1254: TOOL_FAMILY = gnu
1255: TOOL = gnu
1256: TOOL_PATH =
1257: CC_ARCH_SPEC = -mtune=i486 -march=i486
1258: LIBPATH =
1259: LIBS = -lstdc++
1260:

```

```

1261: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1262: endif
1263:
1264: ifeq ($(BUILD_SPEC),PENTIUMdiab_RTP)
1265: VX_CPU_FAMILY = pentium
1266: CPU = PENTIUM
1267: TOOL_FAMILY = diab
1268: TOOL = diab
1269: TOOL_PATH =
1270: CC_ARCH_SPEC = -tPENTIUMLH:rtp
1271: LIBPATH =
1272: LIBS = -lstdlstd
1273:
1274: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1275: endif
1276:
1277: ifeq ($(BUILD_SPEC),PENTIUMgnu_RTP)
1278: VX_CPU_FAMILY = pentium
1279: CPU = PENTIUM
1280: TOOL_FAMILY = gnu
1281: TOOL = gnu
1282: TOOL_PATH =
1283: CC_ARCH_SPEC = -mtune=i486 -march=i486
1284: LIBPATH =
1285: LIBS = -lstdc++
1286:
1287: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1288: endif
1289:
1290: ifeq ($(BUILD_SPEC),PPC32diab_RTP)
1291: VX_CPU_FAMILY = ppc
1292: CPU = PPC32
1293: TOOL_FAMILY = diab
1294: TOOL = diab
1295: TOOL_PATH =
1296: CC_ARCH_SPEC = -tPPCEH:rtp
1297: LIBPATH =
1298: LIBS = -lstdlstd
1299:

```

```

1300: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1301: endif
1302:
1303: ifeq ($(BUILD_SPEC),PPC32gnu_RTP)
1304: VX_CPU_FAMILY = ppc
1305: CPU = PPC32
1306: TOOL_FAMILY = gnu
1307: TOOL = gnu
1308: TOOL_PATH =
1309: CC_ARCH_SPEC = -mhard-float -mstrict-align -mregnames
1310: LIBPATH =
1311: LIBS = -lstdc++
1312:
1313: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1314: endif
1315:
1316: ifeq ($(BUILD_SPEC),PPC32sfdiab_RTP)
1317: VX_CPU_FAMILY = ppc
1318: CPU = PPC32
1319: TOOL_FAMILY = diab
1320: TOOL = sfdiab
1321: TOOL_PATH =
1322: CC_ARCH_SPEC = -tPPCES:rtp
1323: LIBPATH =
1324: LIBS = -lstlstd
1325:
1326: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1327: endif
1328:
1329: ifeq ($(BUILD_SPEC),PPC32sfgnu_RTP)
1330: VX_CPU_FAMILY = ppc
1331: CPU = PPC32
1332: TOOL_FAMILY = gnu
1333: TOOL = sfgnu
1334: TOOL_PATH =
1335: CC_ARCH_SPEC = -msoft-float -mstrict-align -mregnames
1336: LIBPATH =
1337: LIBS = -lstdc++
1338:

```

```

1339:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1340: endif
1341:
1342: ifeq ($(BUILD_SPEC),SH32diab_RTP)
1343: VX_CPU_FAMILY = sh
1344: CPU = SH32
1345: TOOL_FAMILY = diab
1346: TOOL = diab
1347: TOOL_PATH =
1348: CC_ARCH_SPEC = -tSH4EH:rtp -Xunderscore-leading
1349: LIBPATH =
1350: LIBS = -lstdlstd
1351:
1352:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1353: endif
1354:
1355: ifeq ($(BUILD_SPEC),SH32diable_RTP)
1356: VX_CPU_FAMILY = sh
1357: CPU = SH32
1358: TOOL_FAMILY = diab
1359: TOOL = diable
1360: TOOL_PATH =
1361: CC_ARCH_SPEC = -tSH4LH:rtp -Xunderscore-leading
1362: LIBPATH =
1363: LIBS = -lstdlstd
1364:
1365:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1366: endif
1367:
1368: ifeq ($(BUILD_SPEC),SH32gnu_RTP)
1369: VX_CPU_FAMILY = sh
1370: CPU = SH32
1371: TOOL_FAMILY = gnu
1372: TOOL = gnu
1373: TOOL_PATH =
1374: CC_ARCH_SPEC = -m4 -D__sh
1375: LIBPATH =
1376: LIBS = -lstdc++
1377:

```

```

1378: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1379: endif
1380:
1381: ifeq ($(BUILD_SPEC),SH32gnule_RTP)
1382: VX_CPU_FAMILY = sh
1383: CPU = SH32
1384: TOOL_FAMILY = gnu
1385: TOOL = gnule
1386: TOOL_PATH =
1387: CC_ARCH_SPEC = -m4 -ml -Wl,-EL -D__sh
1388: LIBPATH =
1389: LIBS = -lstdc++
1390:
1391: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1392: endif
1393:
1394: ifeq ($(BUILD_SPEC),SIMPENTIUMdiab_RTP)
1395: VX_CPU_FAMILY = simpentium
1396: CPU = SIMPENTIUM
1397: TOOL_FAMILY = diab
1398: TOOL = diab
1399: TOOL_PATH =
1400: CC_ARCH_SPEC = -tX86LH:rtpsim
1401: LIBPATH =
1402: LIBS = -lstdc++
1403:
1404: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1405: endif
1406:
1407: ifeq ($(BUILD_SPEC),SIMPENTIUMgnu_RTP)
1408: VX_CPU_FAMILY = simpentium
1409: CPU = SIMPENTIUM
1410: TOOL_FAMILY = gnu
1411: TOOL = gnu
1412: TOOL_PATH =
1413: CC_ARCH_SPEC = -mtune=i486 -march=i486
1414: LIBPATH =
1415: LIBS = -lstdc++
1416:

```

```

1417:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1418: endif
1419:
1420: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISdiab_RTP)
1421: VX_CPU_FAMILY = simso
1422: CPU = SIMSPARCSOLARIS
1423: TOOL_FAMILY = diab
1424: TOOL = diab
1425: TOOL_PATH =
1426: CC_ARCH_SPEC = -tSPARCFH:rtpsim
1427: LIBPATH =
1428: LIBS = -lstdlstd
1429:
1430:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1431: endif
1432:
1433: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISgnu_RTP)
1434: VX_CPU_FAMILY = simso
1435: CPU = SIMSPARCSOLARIS
1436: TOOL_FAMILY = gnu
1437: TOOL = gnu
1438: TOOL_PATH =
1439: CC_ARCH_SPEC =
1440: LIBPATH =
1441: LIBS = -lstdc++
1442:
1443:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1444: endif
1445:
1446: ifeq ($(BUILD_SPEC),XSCALEdiab_RTP)
1447: VX_CPU_FAMILY = arm
1448: CPU = XSCALE
1449: TOOL_FAMILY = diab
1450: TOOL = diab
1451: TOOL_PATH =
1452: CC_ARCH_SPEC = -tARMXLS:rtp
1453: LIBPATH =
1454: LIBS = -lstdlstd
1455:

```

```

1456: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1457: endif
1458:
1459: ifeq ($(BUILD_SPEC),XSCALEdiabbe_RTP)
1460: VX_CPU_FAMILY = arm
1461: CPU = XSCALE
1462: TOOL_FAMILY = diab
1463: TOOL = diabbe
1464: TOOL_PATH =
1465: CC_ARCH_SPEC = -tARMXES:rtp
1466: LIBPATH =
1467: LIBS = -lstdlstd
1468:
1469: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1470: endif
1471:
1472: ifeq ($(BUILD_SPEC),XSCALEgnu_RTP)
1473: VX_CPU_FAMILY = arm
1474: CPU = XSCALE
1475: TOOL_FAMILY = gnu
1476: TOOL = gnu
1477: TOOL_PATH =
1478: CC_ARCH_SPEC = -txscale
1479: LIBPATH =
1480: LIBS = -lstdc++
1481:
1482: IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1483: endif
1484:
1485: ifeq ($(BUILD_SPEC),XSCALEgnube_RTP)
1486: VX_CPU_FAMILY = arm
1487: CPU = XSCALE
1488: TOOL_FAMILY = gnu
1489: TOOL = gnube
1490: TOOL_PATH =
1491: CC_ARCH_SPEC = -txscalebe -Wa,-EB
1492: LIBPATH =
1493: LIBS = -lstdc++
1494:

```

```

1495:     IDE_INCLUDES      =      -I$(WIND_USR)/h      -I$(WIND_USR)/h/wrn/coreip      -
IC:/Temp/Test/VxWorks/libRegex
1496: endif
1497:
1498:
1499: ifeq ($(BUILD_SPEC),ARMARCH5diab_RTP)
1500: ifeq ($(DEBUG_MODE),1)
1501: DEBUGFLAGS_C-Compiler = -g
1502: else
1503: DEBUGFLAGS_C-Compiler = -XO
1504: endif
1505: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1506: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1507:
1508: endif
1509: ifeq ($(BUILD_SPEC),ARMARCH5diabbe_RTP)
1510: ifeq ($(DEBUG_MODE),1)
1511: DEBUGFLAGS_C-Compiler = -g
1512: else
1513: DEBUGFLAGS_C-Compiler = -XO
1514: endif
1515: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1516: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
1517:
1518: endif
1519: ifeq ($(BUILD_SPEC),ARMARCH5gnu_RTP)
1520: ifeq ($(DEBUG_MODE),1)
1521: DEBUGFLAGS_C-Compiler = -g
1522: else
1523: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1524: endif
1525: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1526: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"
1527:

```



```

1528: endif
1529: ifeq ($(BUILD_SPEC),ARMARCH5gnube_RTP)
1530: ifeq ($(DEBUG_MODE),1)
1531: DEBUGFLAGS_C-Compiler = -g
1532: else
1533: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1534: endif
1535: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1536:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"
1537:
1538: endif
1539: ifeq ($(BUILD_SPEC),ARMARCH6diab_RTP)
1540: ifeq ($(DEBUG_MODE),1)
1541: DEBUGFLAGS_C-Compiler = -g
1542: else
1543: DEBUGFLAGS_C-Compiler = -XO
1544: endif
1545: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1546:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1547:
1548: endif
1549: ifeq ($(BUILD_SPEC),ARMARCH6diabbe_RTP)
1550: ifeq ($(DEBUG_MODE),1)
1551: DEBUGFLAGS_C-Compiler = -g
1552: else
1553: DEBUGFLAGS_C-Compiler = -XO
1554: endif
1555: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1556:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
1557:
1558: endif
1559: ifeq ($(BUILD_SPEC),ARMARCH6gnu_RTP)
1560: ifeq ($(DEBUG_MODE),1)
1561: DEBUGFLAGS_C-Compiler = -g

```

```

1562: else
1563: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1564: endif
1565: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1566:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"
1567:
1568: endif
1569: ifeq ($(BUILD_SPEC),ARMARCH6gnube_RTP)
1570: ifeq ($(DEBUG_MODE),1)
1571: DEBUGFLAGS_C-Compiler = -g
1572: else
1573: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1574: endif
1575: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1576:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"
1577:
1578: endif
1579: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
1580: ifeq ($(DEBUG_MODE),1)
1581: DEBUGFLAGS_C-Compiler = -g
1582: else
1583: DEBUGFLAGS_C-Compiler = -XO
1584: endif
1585: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1586:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
1587:
1588: endif
1589: ifeq ($(BUILD_SPEC),MIPS32sfdiable_RTP)
1590: ifeq ($(DEBUG_MODE),1)
1591: DEBUGFLAGS_C-Compiler = -g
1592: else
1593: DEBUGFLAGS_C-Compiler = -XO
1594: endif

```

```

1595: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1596: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
1597:
1598: endif
1599: ifeq ($(BUILD_SPEC),MIPS32sfgnu_RTP)
1600: ifeq ($(DEBUG_MODE),1)
1601: DEBUGFLAGS_C-Compiler = -g
1602: else
1603: DEBUGFLAGS_C-Compiler = -O2
1604: endif
1605: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1606: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
1607:
1608: endif
1609: ifeq ($(BUILD_SPEC),MIPS32sfgnule_RTP)
1610: ifeq ($(DEBUG_MODE),1)
1611: DEBUGFLAGS_C-Compiler = -g
1612: else
1613: DEBUGFLAGS_C-Compiler = -O2
1614: endif
1615: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1616: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
1617:
1618: endif
1619: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
1620: ifeq ($(DEBUG_MODE),1)
1621: DEBUGFLAGS_C-Compiler = -g
1622: else
1623: DEBUGFLAGS_C-Compiler = -XO
1624: endif
1625: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c

```

```

1626: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
1627:
1628: endif
1629: ifeq ($(BUILD_SPEC),MIPS64diabla_RTP)
1630: ifeq ($(DEBUG_MODE),1)
1631: DEBUGFLAGS_C-Compiler = -g
1632: else
1633: DEBUGFLAGS_C-Compiler = -XO
1634: endif
1635: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1636: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
1637:
1638: endif
1639: ifeq ($(BUILD_SPEC),MIPS64gnu_RTP)
1640: ifeq ($(DEBUG_MODE),1)
1641: DEBUGFLAGS_C-Compiler = -g
1642: else
1643: DEBUGFLAGS_C-Compiler = -O2
1644: endif
1645: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1646: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
-DMIPSEB $(DEFINES) -o "$@" -c "$<"
1647:
1648: endif
1649: ifeq ($(BUILD_SPEC),MIPS64gnule_RTP)
1650: ifeq ($(DEBUG_MODE),1)
1651: DEBUGFLAGS_C-Compiler = -g
1652: else
1653: DEBUGFLAGS_C-Compiler = -O2
1654: endif
1655: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1656: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
-DMIPSEL $(DEFINES) -o "$@" -c "$<"

```

```

1657:
1658: endif
1659: ifeq ($(BUILD_SPEC),PENTIUM2diab_RTP)
1660: ifeq ($(DEBUG_MODE),1)
1661: DEBUGFLAGS_C-Compiler = -g
1662: else
1663: DEBUGFLAGS_C-Compiler = -XO
1664: endif
1665: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1666: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1667:
1668: endif
1669: ifeq ($(BUILD_SPEC),PENTIUM2gnu_RTP)
1670: ifeq ($(DEBUG_MODE),1)
1671: DEBUGFLAGS_C-Compiler = -g
1672: else
1673: DEBUGFLAGS_C-Compiler = -O2 -fno-builtin
1674: endif
1675: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1676: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium $(DEBUGFLAGS_C-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1677:
1678: endif
1679: ifeq ($(BUILD_SPEC),PENTIUM3diab_RTP)
1680: ifeq ($(DEBUG_MODE),1)
1681: DEBUGFLAGS_C-Compiler = -g
1682: else
1683: DEBUGFLAGS_C-Compiler = -XO
1684: endif
1685: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1686: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1687:
1688: endif
1689: ifeq ($(BUILD_SPEC),PENTIUM3gnu_RTP)
1690: ifeq ($(DEBUG_MODE),1)

```

```

1691: DEBUGFLAGS_C-Compiler = -g
1692: else
1693: DEBUGFLAGS_C-Compiler = -O2 -fno-builtin
1694: endif
1695: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1696: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium $(DEBUGFLAGS_C-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1697:
1698: endif
1699: ifeq ($(BUILD_SPEC),PENTIUM4diab_RTP)
1700: ifeq ($(DEBUG_MODE),1)
1701: DEBUGFLAGS_C-Compiler = -g
1702: else
1703: DEBUGFLAGS_C-Compiler = -XO
1704: endif
1705: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1706: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1707:
1708: endif
1709: ifeq ($(BUILD_SPEC),PENTIUM4gnu_RTP)
1710: ifeq ($(DEBUG_MODE),1)
1711: DEBUGFLAGS_C-Compiler = -g
1712: else
1713: DEBUGFLAGS_C-Compiler = -O2 -fno-builtin
1714: endif
1715: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1716: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium $(DEBUGFLAGS_C-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1717:
1718: endif
1719: ifeq ($(BUILD_SPEC),PENTIUMdiab_RTP)
1720: ifeq ($(DEBUG_MODE),1)
1721: DEBUGFLAGS_C-Compiler = -g
1722: else
1723: DEBUGFLAGS_C-Compiler = -XO
1724: endif

```

```

1725: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1726: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1727:
1728: endif
1729: ifeq ($(BUILD_SPEC),PENTIUMgnu_RTP)
1730: ifeq ($(DEBUG_MODE),1)
1731: DEBUGFLAGS_C-Compiler = -g
1732: else
1733: DEBUGFLAGS_C-Compiler = -O2 -fno-builtin
1734: endif
1735: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1736: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium $(DEBUGFLAGS_C-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1737:
1738: endif
1739: ifeq ($(BUILD_SPEC),PPC32diab_RTP)
1740: ifeq ($(DEBUG_MODE),1)
1741: DEBUGFLAGS_C-Compiler = -g
1742: else
1743: DEBUGFLAGS_C-Compiler = -XO
1744: endif
1745: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1746: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1747:
1748: endif
1749: ifeq ($(BUILD_SPEC),PPC32gnu_RTP)
1750: ifeq ($(DEBUG_MODE),1)
1751: DEBUGFLAGS_C-Compiler = -g
1752: else
1753: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1754: endif
1755: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1756: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccppc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)

```

```

$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"
1757:
1758: endif
1759: ifeq ($(BUILD_SPEC),PPC32sfdiab_RTP)
1760: ifeq ($(DEBUG_MODE),1)
1761: DEBUGFLAGS_C-Compiler = -g
1762: else
1763: DEBUGFLAGS_C-Compiler = -XO
1764: endif
1765: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1766: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1767:
1768: endif
1769: ifeq ($(BUILD_SPEC),PPC32sfgnu_RTP)
1770: ifeq ($(DEBUG_MODE),1)
1771: DEBUGFLAGS_C-Compiler = -g
1772: else
1773: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1774: endif
1775: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1776: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccppc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"
1777:
1778: endif
1779: ifeq ($(BUILD_SPEC),SH32diab_RTP)
1780: ifeq ($(DEBUG_MODE),1)
1781: DEBUGFLAGS_C-Compiler = -g
1782: else
1783: DEBUGFLAGS_C-Compiler = -XO
1784: endif
1785: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1786: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
1787:
1788: endif

```



```

1789: ifeq ($(BUILD_SPEC),SH32diab_RTP)
1790: ifeq ($(DEBUG_MODE),1)
1791: DEBUGFLAGS_C-Compiler = -g
1792: else
1793: DEBUGFLAGS_C-Compiler = -XO
1794: endif
1795: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1796: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
1797:
1798: endif
1799: ifeq ($(BUILD_SPEC),SH32gnu_RTP)
1800: ifeq ($(DEBUG_MODE),1)
1801: DEBUGFLAGS_C-Compiler = -g
1802: else
1803: DEBUGFLAGS_C-Compiler = -O2 -fno-builtin
1804: endif
1805: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1806: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccsh $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
-D_SH7750 $(DEFINES) -o "$@" -c "$<"
1807:
1808: endif
1809: ifeq ($(BUILD_SPEC),SH32gnule_RTP)
1810: ifeq ($(DEBUG_MODE),1)
1811: DEBUGFLAGS_C-Compiler = -g
1812: else
1813: DEBUGFLAGS_C-Compiler = -O2 -fno-builtin
1814: endif
1815: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1816: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccsh $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
-D_SH7750 $(DEFINES) -o "$@" -c "$<"
1817:
1818: endif
1819: ifeq ($(BUILD_SPEC),SIMPENTIUMdiab_RTP)
1820: ifeq ($(DEBUG_MODE),1)
1821: DEBUGFLAGS_C-Compiler = -g
1822: else

```

```

1823: DEBUGFLAGS_C-Compiler = -XO
1824: endif
1825: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1826: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1827:
1828: endif
1829: ifeq ($(BUILD_SPEC),SIMPENTIUMgnu_RTP)
1830: ifeq ($(DEBUG_MODE),1)
1831: DEBUGFLAGS_C-Compiler = -g
1832: else
1833: DEBUGFLAGS_C-Compiler = -O2 -fno-builtin -fno-defer-pop
1834: endif
1835: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1836: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium $(DEBUGFLAGS_C-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1837:
1838: endif
1839: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISdiab_RTP)
1840: ifeq ($(DEBUG_MODE),1)
1841: DEBUGFLAGS_C-Compiler = -g
1842: else
1843: DEBUGFLAGS_C-Compiler = -XO
1844: endif
1845: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1846: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1847:
1848: endif
1849: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISgnu_RTP)
1850: ifeq ($(DEBUG_MODE),1)
1851: DEBUGFLAGS_C-Compiler = -g
1852: else
1853: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1854: endif
1855: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c

```

```

1856: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccsparc $(DEBUGFLAGS_C-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1857:
1858: endif
1859: ifeq ($(BUILD_SPEC),XSCALEdiab_RTP)
1860: ifeq ($(DEBUG_MODE),1)
1861: DEBUGFLAGS_C-Compiler = -g
1862: else
1863: DEBUGFLAGS_C-Compiler = -XO
1864: endif
1865: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1866: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1867:
1868: endif
1869: ifeq ($(BUILD_SPEC),XSCALEdiabbe_RTP)
1870: ifeq ($(DEBUG_MODE),1)
1871: DEBUGFLAGS_C-Compiler = -g
1872: else
1873: DEBUGFLAGS_C-Compiler = -XO
1874: endif
1875: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1876: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
1877:
1878: endif
1879: ifeq ($(BUILD_SPEC),XSCALEgnu_RTP)
1880: ifeq ($(DEBUG_MODE),1)
1881: DEBUGFLAGS_C-Compiler = -g
1882: else
1883: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1884: endif
1885: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1886: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"

```

```

1887:
1888: endif
1889: ifeq ($(BUILD_SPEC),XSCALEgnube_RTP)
1890: ifeq ($(DEBUG_MODE),1)
1891: DEBUGFLAGS_C-Compiler = -g
1892: else
1893: DEBUGFLAGS_C-Compiler = -O2 -fstrength-reduce -fno-builtin
1894: endif
1895: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.c
1896: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_C-Compiler)
$(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_CFLAGS) $(IDE_INCLUDES)
$(ADDED_INCLUDES) -DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL)
$(DEFINES) -o "$@" -c "$<"
1897:
1898: endif
1899: ifeq ($(BUILD_SPEC),ARMARCH5diab_RTP)
1900: ifeq ($(DEBUG_MODE),1)
1901: DEBUGFLAGS_C++-Compiler = -g
1902: else
1903: DEBUGFLAGS_C++-Compiler = -XO
1904: endif
1905: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
1906: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1907:
1908: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
1909: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1910:
1911: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
1912: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1913:
1914: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
1915: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"

```

```

1916:
1917: endif
1918: ifeq ($(BUILD_SPEC),ARMARCH5diabbe_RTP)
1919: ifeq ($(DEBUG_MODE),1)
1920: DEBUGFLAGS_C++-Compiler = -g
1921: else
1922: DEBUGFLAGS_C++-Compiler = -XO
1923: endif
1924: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
1925: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
1926:
1927: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
1928: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
1929:
1930: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
1931: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
1932:
1933: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
1934: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
1935:
1936: endif
1937: ifeq ($(BUILD_SPEC),ARMARCH5gnu_RTP)
1938: ifeq ($(DEBUG_MODE),1)
1939: DEBUGFLAGS_C++-Compiler = -g
1940: else
1941: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
1942: endif
1943: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
1944: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"

```

```

1945:
1946: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
1947: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1948:
1949: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
1950: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1951:
1952: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
1953: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1954:
1955: endif
1956: ifeq ($(BUILD_SPEC),ARMARCH5gnube_RTP)
1957: ifeq ($(DEBUG_MODE),1)
1958: DEBUGFLAGS_C++-Compiler = -g
1959: else
1960: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
1961: endif
1962: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
1963: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1964:
1965: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
1966: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1967:
1968: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
1969: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1970:
1971: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc

```

```

1972: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
1973:
1974: endif
1975: ifeq ($(BUILD_SPEC),ARMARCH6diab_RTP)
1976: ifeq ($(DEBUG_MODE),1)
1977: DEBUGFLAGS_C++-Compiler = -g
1978: else
1979: DEBUGFLAGS_C++-Compiler = -XO
1980: endif
1981: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
1982: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1983:
1984: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
1985: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1986:
1987: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
1988: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1989:
1990: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
1991: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
1992:
1993: endif
1994: ifeq ($(BUILD_SPEC),ARMARCH6diabbe_RTP)
1995: ifeq ($(DEBUG_MODE),1)
1996: DEBUGFLAGS_C++-Compiler = -g
1997: else
1998: DEBUGFLAGS_C++-Compiler = -XO
1999: endif
2000: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp

```

```

2001: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2002:
2003: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2004: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2005:
2006: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2007: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2008:
2009: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2010: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2011:
2012: endif
2013: ifeq ($(BUILD_SPEC),ARMARCH6gnu_RTP)
2014: ifeq ($(DEBUG_MODE),1)
2015: DEBUGFLAGS_C++-Compiler = -g
2016: else
2017: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
2018: endif
2019: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2020: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2021:
2022: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2023: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2024:
2025: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2026: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)

```



```

$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2027:
2028: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2029: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2030:
2031: endif
2032: ifeq ($(BUILD_SPEC),ARMARCH6gnube_RTP)
2033: ifeq ($(DEBUG_MODE),1)
2034: DEBUGFLAGS_C++-Compiler = -g
2035: else
2036: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
2037: endif
2038: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2039: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2040:
2041: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2042: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2043:
2044: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2045: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2046:
2047: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2048: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2049:
2050: endif
2051: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
2052: ifeq ($(DEBUG_MODE),1)
2053: DEBUGFLAGS_C++-Compiler = -g

```

```

2054: else
2055: DEBUGFLAGS_C++-Compiler = -XO
2056: endif
2057: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2058: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2059:
2060: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2061: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2062:
2063: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2064: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2065:
2066: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2067: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2068:
2069: endif
2070: ifeq ($(BUILD_SPEC),MIPS32sfdisable_RTP)
2071: ifeq ($(DEBUG_MODE),1)
2072: DEBUGFLAGS_C++-Compiler = -g
2073: else
2074: DEBUGFLAGS_C++-Compiler = -XO
2075: endif
2076: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2077: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2078:

```

```

2079: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2080: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2081:
2082: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2083: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2084:
2085: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2086: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2087:
2088: endif
2089: ifeq ($(BUILD_SPEC),MIPS32sfgnu_RTP)
2090: ifeq ($(DEBUG_MODE),1)
2091: DEBUGFLAGS_C++-Compiler = -g
2092: else
2093: DEBUGFLAGS_C++-Compiler = -O2
2094: endif
2095: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2096: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2097:
2098: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2099: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2100:
2101: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2102: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)

```

```

$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2103:
2104: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2105: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2106:
2107: endif
2108: ifeq ($(BUILD_SPEC),MIPS32sfgnule_RTP)
2109: ifeq ($(DEBUG_MODE),1)
2110: DEBUGFLAGS_C++-Compiler = -g
2111: else
2112: DEBUGFLAGS_C++-Compiler = -O2
2113: endif
2114: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2115: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2116:
2117: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2118: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2119:
2120: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2121: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2122:
2123: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2124: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -msoft-float -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"

```

```

2125:
2126: endif
2127: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
2128: ifeq ($(DEBUG_MODE),1)
2129: DEBUGFLAGS_C++-Compiler = -g
2130: else
2131: DEBUGFLAGS_C++-Compiler = -XO
2132: endif
2133: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2134: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2135:
2136: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2137: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2138:
2139: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2140: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2141:
2142: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2143: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2144:
2145: endif
2146: ifeq ($(BUILD_SPEC),MIPS64diable_RTP)
2147: ifeq ($(DEBUG_MODE),1)
2148: DEBUGFLAGS_C++-Compiler = -g
2149: else
2150: DEBUGFLAGS_C++-Compiler = -XO
2151: endif
2152: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2153: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"

```

```

2154:
2155: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2156: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2157:
2158: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2159: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2160:
2161: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2162: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2163:
2164: endif
2165: ifeq ($(BUILD_SPEC),MIPS64gnu_RTP)
2166: ifeq ($(DEBUG_MODE),1)
2167: DEBUGFLAGS_C++-Compiler = -g
2168: else
2169: DEBUGFLAGS_C++-Compiler = -O2
2170: endif
2171: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2172: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2173:
2174: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2175: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2176:
2177: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2178: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2179:
2180: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc

```

```

2181: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2182:
2183: endif
2184: ifeq ($(BUILD_SPEC),MIPS64gnule_RTP)
2185: ifeq ($(DEBUG_MODE),1)
2186: DEBUGFLAGS_C++-Compiler = -g
2187: else
2188: DEBUGFLAGS_C++-Compiler = -O2
2189: endif
2190: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2191: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2192:
2193: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2194: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2195:
2196: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2197: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2198:
2199: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2200: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++mips $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2201:
2202: endif
2203: ifeq ($(BUILD_SPEC),PENTIUM2diab_RTP)
2204: ifeq ($(DEBUG_MODE),1)
2205: DEBUGFLAGS_C++-Compiler = -g
2206: else
2207: DEBUGFLAGS_C++-Compiler = -XO
2208: endif
2209: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp

```

```

2210: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2211:
2212: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2213: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2214:
2215: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2216: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2217:
2218: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2219: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2220:
2221: endif
2222: ifeq ($(BUILD_SPEC),PENTIUM2gnu_RTP)
2223: ifeq ($(DEBUG_MODE),1)
2224: DEBUGFLAGS_C++-Compiler = -g
2225: else
2226: DEBUGFLAGS_C++-Compiler = -O2 -fno-builtin
2227: endif
2228: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2229: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2230:
2231: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2232: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2233:
2234: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2235: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)

```



```

$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2236:
2237: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2238: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2239:
2240: endif
2241: ifeq ($(BUILD_SPEC),PENTIUM3diab_RTP)
2242: ifeq ($(DEBUG_MODE),1)
2243: DEBUGFLAGS_C++-Compiler = -g
2244: else
2245: DEBUGFLAGS_C++-Compiler = -XO
2246: endif
2247: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2248: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2249:
2250: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2251: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2252:
2253: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2254: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2255:
2256: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2257: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2258:
2259: endif
2260: ifeq ($(BUILD_SPEC),PENTIUM3gnu_RTP)
2261: ifeq ($(DEBUG_MODE),1)
2262: DEBUGFLAGS_C++-Compiler = -g

```

```

2263: else
2264: DEBUGFLAGS_C++-Compiler = -O2 -fno-builtin
2265: endif
2266: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2267: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2268:
2269: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2270: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2271:
2272: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2273: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2274:
2275: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2276: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2277:
2278: endif
2279: ifeq ($(BUILD_SPEC),PENTIUM4diab_RTP)
2280: ifeq ($(DEBUG_MODE),1)
2281: DEBUGFLAGS_C++-Compiler = -g
2282: else
2283: DEBUGFLAGS_C++-Compiler = -XO
2284: endif
2285: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2286: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2287:
2288: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2289: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"

```

```

2290:
2291: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2292: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2293:
2294: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2295: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2296:
2297: endif
2298: ifeq ($(BUILD_SPEC),PENTIUM4gnu_RTP)
2299: ifeq ($(DEBUG_MODE),1)
2300: DEBUGFLAGS_C++-Compiler = -g
2301: else
2302: DEBUGFLAGS_C++-Compiler = -O2 -fno-builtin
2303: endif
2304: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2305: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2306:
2307: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2308: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2309:
2310: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2311: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2312:
2313: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2314: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2315:
2316: endif

```

```

2317: ifeq ($(BUILD_SPEC),PENTIUMdiab_RTP)
2318: ifeq ($(DEBUG_MODE),1)
2319: DEBUGFLAGS_C++-Compiler = -g
2320: else
2321: DEBUGFLAGS_C++-Compiler = -XO
2322: endif
2323: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2324: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2325:
2326: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2327: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2328:
2329: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2330: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2331:
2332: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2333: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2334:
2335: endif
2336: ifeq ($(BUILD_SPEC),PENTIUMgnu_RTP)
2337: ifeq ($(DEBUG_MODE),1)
2338: DEBUGFLAGS_C++-Compiler = -g
2339: else
2340: DEBUGFLAGS_C++-Compiler = -O2 -fno-builtin
2341: endif
2342: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2343: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2344:
2345: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C

```

```

2346: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2347:
2348: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2349: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2350:
2351: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2352: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2353:
2354: endif
2355: ifeq ($(BUILD_SPEC),PPC32diab_RTP)
2356: ifeq ($(DEBUG_MODE),1)
2357: DEBUGFLAGS_C++-Compiler = -g
2358: else
2359: DEBUGFLAGS_C++-Compiler = -XO
2360: endif
2361: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2362: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2363:
2364: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2365: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2366:
2367: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2368: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2369:
2370: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2371: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic

```

```

$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2372:
2373: endif
2374: ifeq ($(BUILD_SPEC),PPC32gnu_RTP)
2375: ifeq ($(DEBUG_MODE),1)
2376: DEBUGFLAGS_C++-Compiler = -g
2377: else
2378: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
2379: endif
2380: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2381: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2382:
2383: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2384: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2385:
2386: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2387: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2388:
2389: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2390: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2391:
2392: endif
2393: ifeq ($(BUILD_SPEC),PPC32sfdiab_RTP)
2394: ifeq ($(DEBUG_MODE),1)
2395: DEBUGFLAGS_C++-Compiler = -g
2396: else
2397: DEBUGFLAGS_C++-Compiler = -XO
2398: endif
2399: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2400: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic

```

```

$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2401:
2402: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2403: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2404:
2405: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2406: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2407:
2408: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2409: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2410:
2411: endif
2412: ifeq ($(BUILD_SPEC),PPC32sfgnu_RTP)
2413: ifeq ($(DEBUG_MODE),1)
2414: DEBUGFLAGS_C++-Compiler = -g
2415: else
2416: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
2417: endif
2418: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2419: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2420:
2421: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2422: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2423:
2424: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2425: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"

```

```

2426:
2427: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2428: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++ppc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2429:
2430: endif
2431: ifeq ($(BUILD_SPEC),SH32diab_RTP)
2432: ifeq ($(DEBUG_MODE),1)
2433: DEBUGFLAGS_C++-Compiler = -g
2434: else
2435: DEBUGFLAGS_C++-Compiler = -XO
2436: endif
2437: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2438: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2439:
2440: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2441: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2442:
2443: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2444: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2445:
2446: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2447: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2448:
2449: endif
2450: ifeq ($(BUILD_SPEC),SH32diab_RTP)
2451: ifeq ($(DEBUG_MODE),1)
2452: DEBUGFLAGS_C++-Compiler = -g
2453: else
2454: DEBUGFLAGS_C++-Compiler = -XO

```



```

2455: endif
2456: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2457: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2458:
2459: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2460: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2461:
2462: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2463: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2464:
2465: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2466: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2467:
2468: endif
2469: ifeq ($(BUILD_SPEC),SH32gnu_RTP)
2470: ifeq ($(DEBUG_MODE),1)
2471: DEBUGFLAGS_C++-Compiler = -g
2472: else
2473: DEBUGFLAGS_C++-Compiler = -O2 -fno-builtin
2474: endif
2475: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2476: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2477:
2478: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2479: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2480:
2481: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx

```

```

2482: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2483:
2484: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2485: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2486:
2487: endif
2488: ifeq ($(BUILD_SPEC),SH32gnule_RTP)
2489: ifeq ($(DEBUG_MODE),1)
2490: DEBUGFLAGS_C++-Compiler = -g
2491: else
2492: DEBUGFLAGS_C++-Compiler = -O2 -fno-builtin
2493: endif
2494: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2495: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2496:
2497: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2498: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2499:
2500: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2501: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2502:
2503: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2504: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sh $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2505:
2506: endif
2507: ifeq ($(BUILD_SPEC),SIMPENTIUMdiab_RTP)
2508: ifeq ($(DEBUG_MODE),1)

```

```

2509: DEBUGFLAGS_C++-Compiler = -g
2510: else
2511: DEBUGFLAGS_C++-Compiler = -XO
2512: endif
2513: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2514: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2515:
2516: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2517: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2518:
2519: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2520: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2521:
2522: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2523: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2524:
2525: endif
2526: ifeq ($(BUILD_SPEC),SIMPENTIUMgnu_RTP)
2527: ifeq ($(DEBUG_MODE),1)
2528: DEBUGFLAGS_C++-Compiler = -g
2529: else
2530: DEBUGFLAGS_C++-Compiler = -O2 -fno-builtin -fno-defer-pop
2531: endif
2532: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2533: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2534:
2535: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2536: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)

```

```

$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2537:
2538: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2539: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2540:
2541: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2542: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++pentium $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2543:
2544: endif
2545: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISdiab_RTP)
2546: ifeq ($(DEBUG_MODE),1)
2547: DEBUGFLAGS_C++-Compiler = -g
2548: else
2549: DEBUGFLAGS_C++-Compiler = -XO
2550: endif
2551: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2552: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2553:
2554: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2555: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2556:
2557: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2558: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2559:
2560: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2561: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"

```

```

2562:
2563: endif
2564: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISgnu_RTP)
2565: ifeq ($(DEBUG_MODE),1)
2566: DEBUGFLAGS_C++-Compiler = -g
2567: else
2568: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
2569: endif
2570: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2571: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sparc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2572:
2573: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2574: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sparc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2575:
2576: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2577: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sparc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2578:
2579: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2580: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++sparc $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2581:
2582: endif
2583: ifeq ($(BUILD_SPEC),XSCALEdiab_RTP)
2584: ifeq ($(DEBUG_MODE),1)
2585: DEBUGFLAGS_C++-Compiler = -g
2586: else
2587: DEBUGFLAGS_C++-Compiler = -XO
2588: endif
2589: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2590: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"

```

```

2591:
2592: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2593: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
2594:
2595: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2596: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
2597:
2598: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2599: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
2600:
2601: endif
2602: ifeq ($(BUILD_SPEC),XSCALEdiabbe_RTP)
2603: ifeq ($(DEBUG_MODE),1)
2604: DEBUGFLAGS_C++-Compiler = -g
2605: else
2606: DEBUGFLAGS_C++-Compiler = -XO
2607: endif
2608: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2609: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2610:
2611: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2612: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2613:
2614: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2615: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2616:
2617: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc

```

```

2618: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dplus $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd -Xpic
$(ADDED_C++FLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2619:
2620: endif
2621: ifeq ($(BUILD_SPEC),XSCALEgnu_RTP)
2622: ifeq ($(DEBUG_MODE),1)
2623: DEBUGFLAGS_C++-Compiler = -g
2624: else
2625: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
2626: endif
2627: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp
2628: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2629:
2630: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2631: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2632:
2633: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2634: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2635:
2636: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2637: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2638:
2639: endif
2640: ifeq ($(BUILD_SPEC),XSCALEegnube_RTP)
2641: ifeq ($(DEBUG_MODE),1)
2642: DEBUGFLAGS_C++-Compiler = -g
2643: else
2644: DEBUGFLAGS_C++-Compiler = -O2 -fstrength-reduce -fno-builtin
2645: endif
2646: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cpp

```

```

2647: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2648:
2649: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.C
2650: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2651:
2652: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cxx
2653: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2654:
2655: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.cc
2656: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)c++arm $(DEBUGFLAGS_C++-
Compiler) $(CC_ARCH_SPEC) -ansi -mrtp -Wall -MD -MP -fpic $(ADDED_C++FLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2657:
2658: endif
2659: ifeq ($(BUILD_SPEC),ARMARCH5diab_RTP)
2660: ifeq ($(DEBUG_MODE),1)
2661: DEBUGFLAGS_Assembler = -g
2662: else
2663: DEBUGFLAGS_Assembler = -XO
2664: endif
2665: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2666: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
2667:
2668: endif
2669: ifeq ($(BUILD_SPEC),ARMARCH5diabbe_RTP)
2670: ifeq ($(DEBUG_MODE),1)
2671: DEBUGFLAGS_Assembler = -g
2672: else
2673: DEBUGFLAGS_Assembler = -XO
2674: endif
2675: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s

```



```

2676: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Asembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2677:
2678: endif
2679: ifeq ($(BUILD_SPEC),ARMARCH5gnu_RTP)
2680: ifeq ($(DEBUG_MODE),1)
2681: DEBUGFLAGS_Asembler = -g
2682: else
2683: DEBUGFLAGS_Asembler = -O2 -fstrength-reduce -fno-builtin
2684: endif
2685: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2686: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_Asembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2687:
2688: endif
2689: ifeq ($(BUILD_SPEC),ARMARCH5gnube_RTP)
2690: ifeq ($(DEBUG_MODE),1)
2691: DEBUGFLAGS_Asembler = -g
2692: else
2693: DEBUGFLAGS_Asembler = -O2 -fstrength-reduce -fno-builtin
2694: endif
2695: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2696: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_Asembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2697:
2698: endif
2699: ifeq ($(BUILD_SPEC),ARMARCH6diab_RTP)
2700: ifeq ($(DEBUG_MODE),1)
2701: DEBUGFLAGS_Asembler = -g
2702: else
2703: DEBUGFLAGS_Asembler = -XO
2704: endif
2705: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2706: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Asembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"

```

```

2707:
2708: endif
2709: ifeq ($(BUILD_SPEC),ARMARCH6diabbe_RTP)
2710: ifeq ($(DEBUG_MODE),1)
2711: DEBUGFLAGS_Assembler = -g
2712: else
2713: DEBUGFLAGS_Assembler = -XO
2714: endif
2715: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2716: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
2717:
2718: endif
2719: ifeq ($(BUILD_SPEC),ARMARCH6gnu_RTP)
2720: ifeq ($(DEBUG_MODE),1)
2721: DEBUGFLAGS_Assembler = -g
2722: else
2723: DEBUGFLAGS_Assembler = -O2 -fstrength-reduce -fno-builtin
2724: endif
2725: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2726: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2727:
2728: endif
2729: ifeq ($(BUILD_SPEC),ARMARCH6gnube_RTP)
2730: ifeq ($(DEBUG_MODE),1)
2731: DEBUGFLAGS_Assembler = -g
2732: else
2733: DEBUGFLAGS_Assembler = -O2 -fstrength-reduce -fno-builtin
2734: endif
2735: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2736: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2737:
2738: endif
2739: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
2740: ifeq ($(DEBUG_MODE),1)

```

```

2741: DEBUGFLAGS_Assembler = -g
2742: else
2743: DEBUGFLAGS_Assembler = -XO
2744: endif
2745: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2746: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2747:
2748: endif
2749: ifeq ($(BUILD_SPEC),MIPS32sfdiable_RTP)
2750: ifeq ($(DEBUG_MODE),1)
2751: DEBUGFLAGS_Assembler = -g
2752: else
2753: DEBUGFLAGS_Assembler = -XO
2754: endif
2755: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2756: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2757:
2758: endif
2759: ifeq ($(BUILD_SPEC),MIPS32sfgnu_RTP)
2760: ifeq ($(DEBUG_MODE),1)
2761: DEBUGFLAGS_Assembler = -g
2762: else
2763: DEBUGFLAGS_Assembler = -O2
2764: endif
2765: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2766: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2767:
2768: endif
2769: ifeq ($(BUILD_SPEC),MIPS32sfgnule_RTP)
2770: ifeq ($(DEBUG_MODE),1)
2771: DEBUGFLAGS_Assembler = -g

```

```

2772: else
2773: DEBUGFLAGS_Assembler = -O2
2774: endif
2775: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2776: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL -DSOFT_FLOAT
$(DEFINES) -o "$@" -c "$<"
2777:
2778: endif
2779: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
2780: ifeq ($(DEBUG_MODE),1)
2781: DEBUGFLAGS_Assembler = -g
2782: else
2783: DEBUGFLAGS_Assembler = -XO
2784: endif
2785: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2786: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2787:
2788: endif
2789: ifeq ($(BUILD_SPEC),MIPS64diable_RTP)
2790: ifeq ($(DEBUG_MODE),1)
2791: DEBUGFLAGS_Assembler = -g
2792: else
2793: DEBUGFLAGS_Assembler = -XO
2794: endif
2795: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2796: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2797:
2798: endif
2799: ifeq ($(BUILD_SPEC),MIPS64gnu_RTP)
2800: ifeq ($(DEBUG_MODE),1)
2801: DEBUGFLAGS_Assembler = -g
2802: else
2803: DEBUGFLAGS_Assembler = -O2
2804: endif

```

```

2805: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2806:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_Asembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEB $(DEFINES) -o "$@" -c "$<"
2807:
2808: endif
2809: ifeq ($(BUILD_SPEC),MIPS64gnule_RTP)
2810: ifeq ($(DEBUG_MODE),1)
2811: DEBUGFLAGS_Asembler = -g
2812: else
2813: DEBUGFLAGS_Asembler = -O2
2814: endif
2815: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2816:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccmips $(DEBUGFLAGS_Asembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DMIPSEL $(DEFINES) -o "$@" -c "$<"
2817:
2818: endif
2819: ifeq ($(BUILD_SPEC),PENTIUM2diab_RTP)
2820: ifeq ($(DEBUG_MODE),1)
2821: DEBUGFLAGS_Asembler = -g
2822: else
2823: DEBUGFLAGS_Asembler = -XO
2824: endif
2825: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2826:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Asembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2827:
2828: endif
2829: ifeq ($(BUILD_SPEC),PENTIUM2gnu_RTP)
2830: ifeq ($(DEBUG_MODE),1)
2831: DEBUGFLAGS_Asembler = -g
2832: else
2833: DEBUGFLAGS_Asembler = -O2 -fno-builtin
2834: endif
2835: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2836:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium
$(DEBUGFLAGS_Asembler) $(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -

```

```

fpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2837:
2838: endif
2839: ifeq ($(BUILD_SPEC),PENTIUM3diab_RTP)
2840: ifeq ($(DEBUG_MODE),1)
2841: DEBUGFLAGS_Assembler = -g
2842: else
2843: DEBUGFLAGS_Assembler = -XO
2844: endif
2845: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2846: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2847:
2848: endif
2849: ifeq ($(BUILD_SPEC),PENTIUM3gnu_RTP)
2850: ifeq ($(DEBUG_MODE),1)
2851: DEBUGFLAGS_Assembler = -g
2852: else
2853: DEBUGFLAGS_Assembler = -O2 -fno-builtin
2854: endif
2855: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2856: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium
$(DEBUGFLAGS_Assembler) $(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -
fpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2857:
2858: endif
2859: ifeq ($(BUILD_SPEC),PENTIUM4diab_RTP)
2860: ifeq ($(DEBUG_MODE),1)
2861: DEBUGFLAGS_Assembler = -g
2862: else
2863: DEBUGFLAGS_Assembler = -XO
2864: endif
2865: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2866: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2867:
2868: endif

```

```

2869: ifeq ($(BUILD_SPEC),PENTIUM4gnu_RTP)
2870: ifeq ($(DEBUG_MODE),1)
2871: DEBUGFLAGS_Assembler = -g
2872: else
2873: DEBUGFLAGS_Assembler = -O2 -fno-builtin
2874: endif
2875: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2876:  $(TRACE_FLAG)echo          "building          $@";          $(TOOL_PATH)ccpentium
$(DEBUGFLAGS_Assembler) $(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -
fpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2877:
2878: endif
2879: ifeq ($(BUILD_SPEC),PENTIUMdiab_RTP)
2880: ifeq ($(DEBUG_MODE),1)
2881: DEBUGFLAGS_Assembler = -g
2882: else
2883: DEBUGFLAGS_Assembler = -XO
2884: endif
2885: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2886:  $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2887:
2888: endif
2889: ifeq ($(BUILD_SPEC),PENTIUMgnu_RTP)
2890: ifeq ($(DEBUG_MODE),1)
2891: DEBUGFLAGS_Assembler = -g
2892: else
2893: DEBUGFLAGS_Assembler = -O2 -fno-builtin
2894: endif
2895: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2896:  $(TRACE_FLAG)echo          "building          $@";          $(TOOL_PATH)ccpentium
$(DEBUGFLAGS_Assembler) $(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -
fpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2897:
2898: endif
2899: ifeq ($(BUILD_SPEC),PPC32diab_RTP)
2900: ifeq ($(DEBUG_MODE),1)
2901: DEBUGFLAGS_Assembler = -g
2902: else

```

```

2903: DEBUGFLAGS_Assembler = -XO
2904: endif
2905: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2906: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2907:
2908: endif
2909: ifeq ($(BUILD_SPEC),PPC32gnu_RTP)
2910: ifeq ($(DEBUG_MODE),1)
2911: DEBUGFLAGS_Assembler = -g
2912: else
2913: DEBUGFLAGS_Assembler = -O2 -fstrength-reduce -fno-builtin
2914: endif
2915: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2916: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccppc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2917:
2918: endif
2919: ifeq ($(BUILD_SPEC),PPC32sfdiab_RTP)
2920: ifeq ($(DEBUG_MODE),1)
2921: DEBUGFLAGS_Assembler = -g
2922: else
2923: DEBUGFLAGS_Assembler = -XO
2924: endif
2925: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2926: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2927:
2928: endif
2929: ifeq ($(BUILD_SPEC),PPC32sfgnu_RTP)
2930: ifeq ($(DEBUG_MODE),1)
2931: DEBUGFLAGS_Assembler = -g
2932: else
2933: DEBUGFLAGS_Assembler = -O2 -fstrength-reduce -fno-builtin
2934: endif
2935: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s

```



```

2936: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccppc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2937:
2938: endif
2939: ifeq ($(BUILD_SPEC),SH32diab_RTP)
2940: ifeq ($(DEBUG_MODE),1)
2941: DEBUGFLAGS_Assembler = -g
2942: else
2943: DEBUGFLAGS_Assembler = -XO
2944: endif
2945: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2946: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Wa,-Xalign-power2 -Xmake-
dependency=0xd -Xpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -
DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750
$(DEFINES) -o "$@" -c "$<"
2947:
2948: endif
2949: ifeq ($(BUILD_SPEC),SH32diab_RTP)
2950: ifeq ($(DEBUG_MODE),1)
2951: DEBUGFLAGS_Assembler = -g
2952: else
2953: DEBUGFLAGS_Assembler = -XO
2954: endif
2955: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2956: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Wa,-Xalign-power2 -Xmake-
dependency=0xd -Xpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -
DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750
$(DEFINES) -o "$@" -c "$<"
2957:
2958: endif
2959: ifeq ($(BUILD_SPEC),SH32gnu_RTP)
2960: ifeq ($(DEBUG_MODE),1)
2961: DEBUGFLAGS_Assembler = -g
2962: else
2963: DEBUGFLAGS_Assembler = -O2 -fno-builtin
2964: endif
2965: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2966: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccsh $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)

```

```

$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2967:
2968: endif
2969: ifeq ($(BUILD_SPEC),SH32gnule_RTP)
2970: ifeq ($(DEBUG_MODE),1)
2971: DEBUGFLAGS_Assembler = -g
2972: else
2973: DEBUGFLAGS_Assembler = -O2 -fno-builtin
2974: endif
2975: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2976: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccsh $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES)          $(ADDED_INCLUDES)          -DCPU=$(CPU)          -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -D_SH7750 $(DEFINES) -o "$@" -c "$<"
2977:
2978: endif
2979: ifeq ($(BUILD_SPEC),SIMPENTIUMdiab_RTP)
2980: ifeq ($(DEBUG_MODE),1)
2981: DEBUGFLAGS_Assembler = -g
2982: else
2983: DEBUGFLAGS_Assembler = -XO
2984: endif
2985: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2986: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Wa,-Xmnem-mit -Xmake-
dependency=0xd -Xpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -
DCPU=$(CPU) -DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c
"$<"
2987:
2988: endif
2989: ifeq ($(BUILD_SPEC),SIMPENTIUMgnu_RTP)
2990: ifeq ($(DEBUG_MODE),1)
2991: DEBUGFLAGS_Assembler = -g
2992: else
2993: DEBUGFLAGS_Assembler = -O2 -fno-builtin -fno-defer-pop
2994: endif
2995: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
2996: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccpentium
$(DEBUGFLAGS_Assembler) $(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -
fpic $(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
2997:

```

```

2998: endif
2999: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISdiab_RTP)
3000: ifeq ($(DEBUG_MODE),1)
3001: DEBUGFLAGS_Assembler = -g
3002: else
3003: DEBUGFLAGS_Assembler = -XO
3004: endif
3005: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
3006: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
3007:
3008: endif
3009: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISgnu_RTP)
3010: ifeq ($(DEBUG_MODE),1)
3011: DEBUGFLAGS_Assembler = -g
3012: else
3013: DEBUGFLAGS_Assembler = -O2 -fstrength-reduce -fno-builtin
3014: endif
3015: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
3016: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccsparc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
3017:
3018: endif
3019: ifeq ($(BUILD_SPEC),XSCALEdiab_RTP)
3020: ifeq ($(DEBUG_MODE),1)
3021: DEBUGFLAGS_Assembler = -g
3022: else
3023: DEBUGFLAGS_Assembler = -XO
3024: endif
3025: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
3026: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEL $(DEFINES) -o "$@" -c "$<"
3027:
3028: endif
3029: ifeq ($(BUILD_SPEC),XSCALEdiabbe_RTP)
3030: ifeq ($(DEBUG_MODE),1)
3031: DEBUGFLAGS_Assembler = -g

```

```

3032: else
3033: DEBUGFLAGS_Assembler = -XO
3034: endif
3035: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
3036: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dcc $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -Xansi -Xpreprocess-assembly -Xcpp-no-space -Xmake-dependency=0xd -Xpic
$(ADDED_CFLAGS) $(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) -DARMEB $(DEFINES) -o "$@" -c "$<"
3037:
3038: endif
3039: ifeq ($(BUILD_SPEC),XSCALEgnu_RTP)
3040: ifeq ($(DEBUG_MODE),1)
3041: DEBUGFLAGS_Assembler = -g
3042: else
3043: DEBUGFLAGS_Assembler = -O2 -fstrength-reduce -fno-builtin
3044: endif
3045: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
3046: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
3047:
3048: endif
3049: ifeq ($(BUILD_SPEC),XSCALEgnube_RTP)
3050: ifeq ($(DEBUG_MODE),1)
3051: DEBUGFLAGS_Assembler = -g
3052: else
3053: DEBUGFLAGS_Assembler = -O2 -fstrength-reduce -fno-builtin
3054: endif
3055: $(OBJ_DIR)/%.sho : $(SRC_DIR)/%.s
3056: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ccarm $(DEBUGFLAGS_Assembler)
$(CC_ARCH_SPEC) -ansi -mrtp -P -xassembler-with-cpp -MD -MP -fpic $(ADDED_CFLAGS)
$(IDE_INCLUDES) $(ADDED_INCLUDES) -DCPU=$(CPU) -
DTOOL_FAMILY=$(TOOL_FAMILY) -DTOOL=$(TOOL) $(DEFINES) -o "$@" -c "$<"
3057:
3058: endif
3059:
3060: ifeq ($(BUILD_SPEC),ARMARCH5diab_RTP)
3061: ifeq ($(DEBUG_MODE),1)
3062: DEBUGFLAGS_libRegex =
3063: else
3064: DEBUGFLAGS_libRegex =
3065: endif

```

```

3066: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3067:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3068:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3069:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3070:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3071:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3072:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3073:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3074:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3075:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3076:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3077: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3078: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3079:
3080: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3081: endif
3082: ifeq ($(BUILD_SPEC),ARMARCH5diabbe_RTP)
3083: ifeq ($(DEBUG_MODE),1)
3084: DEBUGFLAGS_libRegex =
3085: else
3086: DEBUGFLAGS_libRegex =
3087: endif
3088: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3089:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3090:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3091:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3092:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3093:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3094:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3095:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3096:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3097:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3098:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3099: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3100: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3101:
3102: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3103: endif
3104: ifeq ($(BUILD_SPEC),ARMARCH5gnu_RTP)

```

```

3105: ifeq ($(DEBUG_MODE),1)
3106: DEBUGFLAGS_libRegex =
3107: else
3108: DEBUGFLAGS_libRegex =
3109: endif
3110: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3111: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3112: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3113: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3114: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3115: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3116: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3117: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3118: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3119: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3120: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3121: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3122: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ar arm crus "$@"
$(OBJECTS_TARGETS_libRegex)
3123:
3124: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3125: endif
3126: ifeq ($(BUILD_SPEC),ARMARCH5gnube_RTP)
3127: ifeq ($(DEBUG_MODE),1)
3128: DEBUGFLAGS_libRegex =
3129: else
3130: DEBUGFLAGS_libRegex =
3131: endif
3132: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3133: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3134: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3135: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3136: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3137: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3138: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3139: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3140: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3141: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3142: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3143: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)

```

```

3144: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)ararm      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3145:
3146: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3147: endif
3148: ifeq ($(BUILD_SPEC),ARMARCH6diab_RTP)
3149: ifeq ($(DEBUG_MODE),1)
3150: DEBUGFLAGS_libRegex =
3151: else
3152: DEBUGFLAGS_libRegex =
3153: endif
3154: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3155: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3156: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3157: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3158: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3159: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3160: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3161: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3162: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3163: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3164: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3165: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3166: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3167:
3168: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3169: endif
3170: ifeq ($(BUILD_SPEC),ARMARCH6diabbe_RTP)
3171: ifeq ($(DEBUG_MODE),1)
3172: DEBUGFLAGS_libRegex =
3173: else
3174: DEBUGFLAGS_libRegex =
3175: endif
3176: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3177: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3178: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3179: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3180: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3181: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3182: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \

```

```

3183: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3184: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3185: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3186: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3187: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3188: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3189:
3190: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3191: endif
3192: ifeq ($(BUILD_SPEC),ARMARCH6gnu_RTP)
3193: ifeq ($(DEBUG_MODE),1)
3194: DEBUGFLAGS_libRegex =
3195: else
3196: DEBUGFLAGS_libRegex =
3197: endif
3198: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3199: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3200: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3201: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3202: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3203: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3204: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3205: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3206: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3207: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3208: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3209: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3210: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ararm crus "$@"
$(OBJECTS_TARGETS_libRegex)
3211:
3212: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3213: endif
3214: ifeq ($(BUILD_SPEC),ARMARCH6gnube_RTP)
3215: ifeq ($(DEBUG_MODE),1)
3216: DEBUGFLAGS_libRegex =
3217: else
3218: DEBUGFLAGS_libRegex =
3219: endif
3220: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3221: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \

```



```

3222: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3223: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3224: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3225: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3226: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3227: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3228: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3229: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3230: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3231: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3232: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ararm crus "$@"
$(OBJECTS_TARGETS_libRegex)
3233:
3234: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3235: endif
3236: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
3237: ifeq ($(DEBUG_MODE),1)
3238: DEBUGFLAGS_libRegex =
3239: else
3240: DEBUGFLAGS_libRegex =
3241: endif
3242: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3243: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3244: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3245: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3246: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3247: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3248: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3249: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3250: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3251: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3252: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3253: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3254: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3255:
3256: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3257: endif
3258: ifeq ($(BUILD_SPEC),MIPS32sfdiab_RTP)
3259: ifeq ($(DEBUG_MODE),1)
3260: DEBUGFLAGS_libRegex =

```

```

3261: else
3262: DEBUGFLAGS_libRegex =
3263: endif
3264: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3265:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3266:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3267:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3268:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3269:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3270:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3271:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3272:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3273:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3274:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3275: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3276: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3277:
3278: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3279: endif
3280: ifeq ($(BUILD_SPEC),MIPS32sfgnu_RTP)
3281: ifeq ($(DEBUG_MODE),1)
3282: DEBUGFLAGS_libRegex =
3283: else
3284: DEBUGFLAGS_libRegex =
3285: endif
3286: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3287:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3288:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3289:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3290:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3291:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3292:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3293:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3294:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3295:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3296:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3297: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3298: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)armips      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3299:

```

```

3300: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3301: endif
3302: ifeq ($(BUILD_SPEC),MIPS32sfgnule_RTP)
3303: ifeq ($(DEBUG_MODE),1)
3304: DEBUGFLAGS_libRegex =
3305: else
3306: DEBUGFLAGS_libRegex =
3307: endif
3308: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3309:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3310:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3311:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3312:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3313:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3314:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3315:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3316:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3317:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3318:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3319: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3320: $(TRACE_FLAG)echo    "building    $@";    $(TOOL_PATH)armips    crus    "$@"
$(OBJECTS_TARGETS_libRegex)
3321:
3322: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3323: endif
3324: ifeq ($(BUILD_SPEC),MIPS64diab_RTP)
3325: ifeq ($(DEBUG_MODE),1)
3326: DEBUGFLAGS_libRegex =
3327: else
3328: DEBUGFLAGS_libRegex =
3329: endif
3330: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3331:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3332:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3333:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3334:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3335:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3336:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3337:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3338:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3339:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \

```

```

3340: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3341: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3342: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3343:
3344: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3345: endif
3346: ifeq ($(BUILD_SPEC),MIPS64diable_RTP)
3347: ifeq ($(DEBUG_MODE),1)
3348: DEBUGFLAGS_libRegex =
3349: else
3350: DEBUGFLAGS_libRegex =
3351: endif
3352: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3353: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3354: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3355: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3356: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3357: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3358: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3359: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3360: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3361: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3362: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3363: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3364: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3365:
3366: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3367: endif
3368: ifeq ($(BUILD_SPEC),MIPS64gnu_RTP)
3369: ifeq ($(DEBUG_MODE),1)
3370: DEBUGFLAGS_libRegex =
3371: else
3372: DEBUGFLAGS_libRegex =
3373: endif
3374: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3375: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3376: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3377: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3378: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \

```

```

3379: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3380: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3381: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3382: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3383: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3384: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3385: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3386: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)armips crus "$@"
$(OBJECTS_TARGETS_libRegex)
3387:
3388: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3389: endif
3390: ifeq ($(BUILD_SPEC),MIPS64gnule_RTP)
3391: ifeq ($(DEBUG_MODE),1)
3392: DEBUGFLAGS_libRegex =
3393: else
3394: DEBUGFLAGS_libRegex =
3395: endif
3396: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3397: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3398: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3399: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3400: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3401: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3402: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3403: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3404: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3405: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3406: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3407: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3408: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)armips crus "$@"
$(OBJECTS_TARGETS_libRegex)
3409:
3410: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3411: endif
3412: ifeq ($(BUILD_SPEC),PENTIUM2diab_RTP)
3413: ifeq ($(DEBUG_MODE),1)
3414: DEBUGFLAGS_libRegex =
3415: else
3416: DEBUGFLAGS_libRegex =
3417: endif

```

```

3418: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3419:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3420:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3421:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3422:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3423:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3424:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3425:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3426:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3427:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3428:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3429: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3430: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3431:
3432: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3433: endif
3434: ifeq ($(BUILD_SPEC),PENTIUM2gnu_RTP)
3435: ifeq ($(DEBUG_MODE),1)
3436: DEBUGFLAGS_libRegex =
3437: else
3438: DEBUGFLAGS_libRegex =
3439: endif
3440: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3441:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3442:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3443:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3444:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3445:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3446:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3447:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3448:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3449:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3450:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3451: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3452: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)arpentium      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3453:
3454: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3455: endif
3456: ifeq ($(BUILD_SPEC),PENTIUM3diab_RTP)

```

```

3457: ifeq ($(DEBUG_MODE),1)
3458: DEBUGFLAGS_libRegex =
3459: else
3460: DEBUGFLAGS_libRegex =
3461: endif
3462: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3463: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3464: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3465: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3466: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3467: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3468: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3469: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3470: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3471: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3472: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3473: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3474: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3475:
3476: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3477: endif
3478: ifeq ($(BUILD_SPEC),PENTIUM3gnu_RTP)
3479: ifeq ($(DEBUG_MODE),1)
3480: DEBUGFLAGS_libRegex =
3481: else
3482: DEBUGFLAGS_libRegex =
3483: endif
3484: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3485: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3486: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3487: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3488: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3489: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3490: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3491: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3492: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3493: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3494: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3495: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)

```

```

3496: $(TRACE_FLAG)echo    "building    $@";    $(TOOL_PATH)arpentium    crus    "$@"
$(OBJECTS_TARGETS_libRegex)
3497:
3498: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3499: endif
3500: ifeq ($(BUILD_SPEC),PENTIUM4diab_RTP)
3501: ifeq ($(DEBUG_MODE),1)
3502: DEBUGFLAGS_libRegex =
3503: else
3504: DEBUGFLAGS_libRegex =
3505: endif
3506: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3507:    $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3508:    $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3509:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3510:    $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3511:    $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3512:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3513:    $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3514:    $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3515:    $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3516:    $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3517: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3518: $(TRACE_FLAG)echo    "building    $@";    $(TOOL_PATH)dar    crus    "$@"
$(OBJECTS_TARGETS_libRegex)
3519:
3520: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3521: endif
3522: ifeq ($(BUILD_SPEC),PENTIUM4gnu_RTP)
3523: ifeq ($(DEBUG_MODE),1)
3524: DEBUGFLAGS_libRegex =
3525: else
3526: DEBUGFLAGS_libRegex =
3527: endif
3528: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3529:    $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3530:    $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3531:    $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3532:    $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3533:    $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3534:    $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \

```



```

3535: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3536: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3537: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3538: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3539: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3540: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)arpentium crus "$@"
$(OBJECTS_TARGETS_libRegex)
3541:
3542: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3543: endif
3544: ifeq ($(BUILD_SPEC),PENTIUMdiab_RTP)
3545: ifeq ($(DEBUG_MODE),1)
3546: DEBUGFLAGS_libRegex =
3547: else
3548: DEBUGFLAGS_libRegex =
3549: endif
3550: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3551: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3552: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3553: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3554: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3555: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3556: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3557: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3558: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3559: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3560: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3561: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3562: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3563:
3564: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3565: endif
3566: ifeq ($(BUILD_SPEC),PENTIUMgnu_RTP)
3567: ifeq ($(DEBUG_MODE),1)
3568: DEBUGFLAGS_libRegex =
3569: else
3570: DEBUGFLAGS_libRegex =
3571: endif
3572: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3573: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \

```

```

3574: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3575: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3576: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3577: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3578: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3579: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3580: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3581: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3582: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3583: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3584: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)arpentium crus "$@"
$(OBJECTS_TARGETS_libRegex)
3585:
3586: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3587: endif
3588: ifeq ($(BUILD_SPEC),PPC32diab_RTP)
3589: ifeq ($(DEBUG_MODE),1)
3590: DEBUGFLAGS_libRegex =
3591: else
3592: DEBUGFLAGS_libRegex =
3593: endif
3594: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3595: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3596: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3597: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3598: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3599: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3600: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3601: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3602: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3603: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3604: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3605: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3606: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3607:
3608: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3609: endif
3610: ifeq ($(BUILD_SPEC),PPC32gnu_RTP)
3611: ifeq ($(DEBUG_MODE),1)
3612: DEBUGFLAGS_libRegex =

```

```

3613: else
3614: DEBUGFLAGS_libRegex =
3615: endif
3616: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3617:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3618:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3619:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3620:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3621:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3622:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3623:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3624:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3625:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3626:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3627: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3628: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)arppc      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3629:
3630: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3631: endif
3632: ifeq ($(BUILD_SPEC),PPC32sfdiab_RTP)
3633: ifeq ($(DEBUG_MODE),1)
3634: DEBUGFLAGS_libRegex =
3635: else
3636: DEBUGFLAGS_libRegex =
3637: endif
3638: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3639:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3640:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3641:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3642:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3643:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3644:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3645:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3646:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3647:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3648:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3649: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3650: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3651:

```

```

3652: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3653: endif
3654: ifeq ($(BUILD_SPEC),PPC32sfgnu_RTP)
3655: ifeq ($(DEBUG_MODE),1)
3656: DEBUGFLAGS_libRegex =
3657: else
3658: DEBUGFLAGS_libRegex =
3659: endif
3660: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3661: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3662: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3663: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3664: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3665: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3666: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3667: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3668: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3669: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3670: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3671: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3672: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)arppc crus "$@"
$(OBJECTS_TARGETS_libRegex)
3673:
3674: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3675: endif
3676: ifeq ($(BUILD_SPEC),SH32diab_RTP)
3677: ifeq ($(DEBUG_MODE),1)
3678: DEBUGFLAGS_libRegex =
3679: else
3680: DEBUGFLAGS_libRegex =
3681: endif
3682: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3683: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3684: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3685: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3686: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3687: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3688: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3689: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3690: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3691: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \

```

```

3692: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3693: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3694: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3695:
3696: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3697: endif
3698: ifeq ($(BUILD_SPEC),SH32diable_RTP)
3699: ifeq ($(DEBUG_MODE),1)
3700: DEBUGFLAGS_libRegex =
3701: else
3702: DEBUGFLAGS_libRegex =
3703: endif
3704: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3705: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3706: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3707: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3708: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3709: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3710: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3711: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3712: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3713: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3714: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3715: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3716: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3717:
3718: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3719: endif
3720: ifeq ($(BUILD_SPEC),SH32gnu_RTP)
3721: ifeq ($(DEBUG_MODE),1)
3722: DEBUGFLAGS_libRegex =
3723: else
3724: DEBUGFLAGS_libRegex =
3725: endif
3726: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3727: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3728: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3729: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3730: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \

```

```

3731: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3732: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3733: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3734: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3735: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3736: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3737: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3738: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)arsh      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3739:
3740: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3741: endif
3742: ifeq ($(BUILD_SPEC),SH32gnule_RTP)
3743: ifeq ($(DEBUG_MODE),1)
3744: DEBUGFLAGS_libRegex =
3745: else
3746: DEBUGFLAGS_libRegex =
3747: endif
3748: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3749: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3750: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3751: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3752: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3753: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3754: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3755: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3756: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3757: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3758: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3759: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3760: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)arsh      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3761:
3762: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3763: endif
3764: ifeq ($(BUILD_SPEC),SIMPENTIUMdiab_RTP)
3765: ifeq ($(DEBUG_MODE),1)
3766: DEBUGFLAGS_libRegex =
3767: else
3768: DEBUGFLAGS_libRegex =
3769: endif

```

```

3770: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3771:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3772:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3773:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3774:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3775:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3776:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3777:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3778:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3779:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3780:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3781: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3782: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3783:
3784: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3785: endif
3786: ifeq ($(BUILD_SPEC),SIMPENTIUMgnu_RTP)
3787: ifeq ($(DEBUG_MODE),1)
3788: DEBUGFLAGS_libRegex =
3789: else
3790: DEBUGFLAGS_libRegex =
3791: endif
3792: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3793:  $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3794:  $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3795:  $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3796:  $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3797:  $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3798:  $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3799:  $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3800:  $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3801:  $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3802:  $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3803: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3804: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)arpentium      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3805:
3806: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3807: endif
3808: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISdiab_RTP)

```

```

3809: ifeq ($(DEBUG_MODE),1)
3810: DEBUGFLAGS_libRegex =
3811: else
3812: DEBUGFLAGS_libRegex =
3813: endif
3814: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3815: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3816: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3817: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3818: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3819: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3820: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3821: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3822: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3823: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3824: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3825: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3826: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3827:
3828: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3829: endif
3830: ifeq ($(BUILD_SPEC),SIMSPARCSOLARISgnu_RTP)
3831: ifeq ($(DEBUG_MODE),1)
3832: DEBUGFLAGS_libRegex =
3833: else
3834: DEBUGFLAGS_libRegex =
3835: endif
3836: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3837: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3838: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3839: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3840: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3841: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3842: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3843: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3844: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3845: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3846: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3847: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)

```



```

3848: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)arsparc      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3849:
3850: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3851: endif
3852: ifeq ($(BUILD_SPEC),XSCALEdiab_RTP)
3853: ifeq ($(DEBUG_MODE),1)
3854: DEBUGFLAGS_libRegex =
3855: else
3856: DEBUGFLAGS_libRegex =
3857: endif
3858: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3859: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3860: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3861: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3862: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3863: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3864: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3865: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3866: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3867: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3868: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3869: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3870: $(TRACE_FLAG)echo      "building      $@";      $(TOOL_PATH)dar      crus      "$@"
$(OBJECTS_TARGETS_libRegex)
3871:
3872: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3873: endif
3874: ifeq ($(BUILD_SPEC),XSCALEdiabbe_RTP)
3875: ifeq ($(DEBUG_MODE),1)
3876: DEBUGFLAGS_libRegex =
3877: else
3878: DEBUGFLAGS_libRegex =
3879: endif
3880: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3881: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3882: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3883: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3884: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3885: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3886: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \

```

```

3887: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3888: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3889: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3890: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3891: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3892: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)dar crus "$@"
$(OBJECTS_TARGETS_libRegex)
3893:
3894: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3895: endif
3896: ifeq ($(BUILD_SPEC),XSCALEgnu_RTP)
3897: ifeq ($(DEBUG_MODE),1)
3898: DEBUGFLAGS_libRegex =
3899: else
3900: DEBUGFLAGS_libRegex =
3901: endif
3902: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3903: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \
3904: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3905: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3906: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3907: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3908: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3909: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3910: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3911: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3912: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3913: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3914: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ararm crus "$@"
$(OBJECTS_TARGETS_libRegex)
3915:
3916: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3917: endif
3918: ifeq ($(BUILD_SPEC),XSCALEgnube_RTP)
3919: ifeq ($(DEBUG_MODE),1)
3920: DEBUGFLAGS_libRegex =
3921: else
3922: DEBUGFLAGS_libRegex =
3923: endif
3924: OBJECTS_TARGETS_libRegex = $(OBJ_DIR)/pcre_get.sho $(OBJ_DIR)/pcre_ucd.sho \
3925: $(OBJ_DIR)/pcre_ord2utf8.sho $(OBJ_DIR)/pcre_newline.sho \

```

```

3926: $(OBJ_DIR)/pcre_xclass.sho $(OBJ_DIR)/pcre_chartables.sho \
3927: $(OBJ_DIR)/pcre_globals.sho $(OBJ_DIR)/pcre_compile.sho \
3928: $(OBJ_DIR)/pcre_exec.sho $(OBJ_DIR)/pcre_info.sho \
3929: $(OBJ_DIR)/pcrecpp.sho $(OBJ_DIR)/pcre_config.sho \
3930: $(OBJ_DIR)/pcre_dfa_exec.sho $(OBJ_DIR)/pcre_refcount.sho \
3931: $(OBJ_DIR)/pcreposix.sho $(OBJ_DIR)/pcre_tables.sho \
3932: $(OBJ_DIR)/pcre_maketables.sho $(OBJ_DIR)/pcre_version.sho \
3933: $(OBJ_DIR)/pcre_try_flipped.sho $(OBJ_DIR)/pcre_study.sho \
3934: $(OBJ_DIR)/pcre_fullinfo.sho $(OBJ_DIR)/pcre_valid_utf8.sho
3935: $(OBJ_DIR)/libRegex.a : $(OBJECTS_TARGETS_libRegex)
3936: $(TRACE_FLAG)echo "building $@"; $(TOOL_PATH)ararm crus "$@"
$(OBJECTS_TARGETS_libRegex)
3937:
3938: target_libRegex : check_objectdir $(OBJ_DIR)/libRegex.a
3939: endif
3940:
3941:
3942:
3943: WIND_SCOPETOOLS_BASE := $(subst \,/,$(WIND_SCOPETOOLS_BASE))
3944:
3945:
3946: -include *.makefile
3947:
3948: main_all : check_objectdir external_build $(PROJECT_TARGETS)
3949: @echo "make: built targets of `pwd`"
3950:
3951: check_objectdir :
3952: @if [ ! -d "$(OBJ_DIR)" ]; then \
3953:     mkdir -p $(OBJ_DIR); \
3954: fi
3955:
3956: # entry point for extending the build
3957: external_build ::
3958: @echo ""
3959:
3960: # main entry point for pre processing prior to the recursion
3961: pre_recursion ::
3962: @echo ""
3963:
3964: # main entry point for post processing after the recursion
3965: post_recursion ::

```

```

3966: @echo ""
3967:
3968: # main entry point for pre processing prior to the build
3969: pre_build :: $(PRE_BUILD_STEP) generate_sources
3970: @echo ""
3971:
3972: # entry point for generating sources prior to the build
3973: generate_sources ::
3974: @echo ""
3975:
3976: # main entry point for post processing after the build
3977: post_build :: $(POST_BUILD_STEP) deploy_output
3978: @echo ""
3979:
3980: # entry point for deploying output after the build
3981: deploy_output ::
3982: @echo ""
3983:
3984: # recursive make in SUBDIRS
3985: subdirs_all :
3986: @_PWD=`pwd`; \
3987: for dir in _dummy_ $(SUBDIRS); do \
3988:     if [ "$$dir" = "_dummy_" ]; then \
3989:         continue ; \
3990:     fi; \
3991:     if [ ! -d "$$dir" ]; then \
3992:         continue ; \
3993:     fi; \
3994:     echo "Recursive make: Changing to Directory '$$dir'; \
3995:     cd "$$dir"; \
3996:     "$(MAKE)" -f "$(MAKEFILE)" $(MFLAGS) all || exit; \
3997:     echo "Recursive make: Changing back to Directory '$$_PWD'; \
3998:     cd "$$_PWD"; \
3999: done
4000:
4001: clean ::
4002: @_PWD=`pwd`; \
4003: for dir in _dummy_ $(SUBDIRS); do \
4004:     if [ "$$dir" = "_dummy_" ]; then \
4005:         continue ; \
4006:     fi; \

```

```

4007:     if [ ! -d "$$dir" ]; then \
4008:         continue ; \
4009:     fi; \
4010:     echo "Recursive make: Changing to Directory '$$dir'; \
4011:     cd "$$dir"; \
4012:     "$$(MAKE)" -f "$$(MAKEFILE)" $(MFLAGS) $@ || exit; \
4013:     echo "Recursive make: Changing back to Directory '$$_PWD'; \
4014:     cd "$$_PWD"; \
4015: done
4016:
4017: clean :: external_clean $(CLEAN_STEP) _clean
4018:
4019: # entry point for extending the build clean
4020: external_clean ::
4021:     @echo ""
4022:
4023: _clean :
4024:     @echo "make: removing targets and objects of `pwd`; \
4025:     rm -f $(OBJECTS) $(PROJECT_TARGETS) $(DEP_FILES) $(wildcard
$(OBJ_DIR)/*.unstripped) $(wildcard $(OBJ_DIR)/ctdt.*)
4026:
4027: build_all_specs :
4028:     @echo "building target default for ALL build-specs"; \
4029:     for spec in _dummy_ $(ALL_BUILD_SPECS); do \
4030:         if [ "$$spec" = "_dummy_" ]; then \
4031:             continue ; \
4032:         fi; \
4033:         echo " "; \
4034:         echo "building all for build-spec '$$spec'; \
4035:         "$$(MAKE)" -f "$$(MAKEFILE)" $(MFLAGS) BUILD_SPEC=$$spec
DEBUG_MODE=$(DEBUG_MODE) TRACE=$(TRACE) || exit; \
4036:     done
4037:
4038: clean_all_specs :
4039:     @echo "building target clean for ALL build-specs"; \
4040:     for spec in _dummy_ $(ALL_BUILD_SPECS); do \
4041:         if [ "$$spec" = "_dummy_" ]; then \
4042:             continue ; \
4043:         fi; \
4044:         echo " "; \
4045:         echo "building clean for build-spec '$$spec'; \

```

```

4046:    "$$(MAKE)"      -f      "$$(MAKEFILE)"      $(MFLAGS)      BUILD_SPEC=$$spec
DEBUG_MODE=$(DEBUG_MODE) TRACE=$(TRACE) clean || exit; \
4047: done
4048:
4049: build_enabled_specs :
4050: @echo "building target default for ENABLED build-specs"; \
4051: for spec in _dummy_ $(ENABLED_BUILD_SPECS); do \
4052:     if [ "$$spec" = "_dummy_" ]; then \
4053:         continue ; \
4054:     fi; \
4055:     echo " "; \
4056:     echo "building all for build-spec '$$spec'"; \
4057:     "$$(MAKE)"      -f      "$$(MAKEFILE)"      $(MFLAGS)      BUILD_SPEC=$$spec
DEBUG_MODE=$(DEBUG_MODE) TRACE=$(TRACE) || exit; \
4058: done
4059:
4060: clean_enabled_specs :
4061: @echo "building target clean for ENABLED build-specs"; \
4062: for spec in _dummy_ $(ENABLED_BUILD_SPECS); do \
4063:     if [ "$$spec" = "_dummy_" ]; then \
4064:         continue ; \
4065:     fi; \
4066:     echo " "; \
4067:     echo "building clean for build-spec '$$spec'"; \
4068:     "$$(MAKE)"      -f      "$$(MAKEFILE)"      $(MFLAGS)      BUILD_SPEC=$$spec
DEBUG_MODE=$(DEBUG_MODE) TRACE=$(TRACE) clean || exit; \
4069: done

```

File: sdm/VxWorks/libRegex/pcreposix.h

```
1: /*****
2: *   Perl-Compatible Regular Expressions   *
3: *****/
4:
5: #ifndef _PCREPOSIX_H
6: #define _PCREPOSIX_H
7:
8: /* This is the header for the POSIX wrapper interface to the PCRE Perl-
9: Compatible Regular Expression library. It defines the things POSIX says should
10: be there. I hope.
11:
12:      Copyright (c) 1997-2008 University of Cambridge
13:
14: -----
15: Redistribution and use in source and binary forms, with or without
16: modification, are permitted provided that the following conditions are met:
17:
18:  * Redistributions of source code must retain the above copyright notice,
19:   this list of conditions and the following disclaimer.
20:
21:  * Redistributions in binary form must reproduce the above copyright
22:   notice, this list of conditions and the following disclaimer in the
23:   documentation and/or other materials provided with the distribution.
24:
25:  * Neither the name of the University of Cambridge nor the names of its
26:   contributors may be used to endorse or promote products derived from
27:   this software without specific prior written permission.
28:
29: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
30: "AS IS"
31: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
32: THE
33: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
34: PURPOSE
35: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
36: BE
37: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
38: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
39: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
```

```

36: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
IN
37: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
38: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
39: POSSIBILITY OF SUCH DAMAGE.
40: -----
41: */
42:
43: /* Have to include stdlib.h in order to ensure that size_t is defined. */
44:
45: #include <stdlib.h>
46:
47: /* Allow for C++ users */
48:
49: #ifdef __cplusplus
50: extern "C" {
51: #endif
52:
53: /* Options, mostly defined by POSIX, but with a couple of extras. */
54:
55: #define REG_ICASE    0x0001
56: #define REG_NEWLINE 0x0002
57: #define REG_NOTBOL   0x0004
58: #define REG_NOTEOL   0x0008
59: #define REG_DOTALL   0x0010 /* NOT defined by POSIX. */
60: #define REG_NOSUB     0x0020
61: #define REG_UTF8      0x0040 /* NOT defined by POSIX. */
62: #define REG_STARTEND 0x0080 /* BSD feature: pass subject string by so, eo */
63:
64: /* This is not used by PCRE, but by defining it we make it easier
65: to slot PCRE into existing programs that make POSIX calls. */
66:
67: #define REG_EXTENDED 0
68:
69: /* Error values. Not all these are relevant or used by the wrapper. */
70:
71: enum {
72:  REG_ASSERT = 1, /* internal error ? */
73:  REG_BADBR,      /* invalid repeat counts in { } */
74:  REG_BADPAT,      /* pattern error */
75:  REG_BADRPT,      /* ? * + invalid */

```



```

76: REG_EBRACE,    /* unbalanced {} */
77: REG_EBRACK,    /* unbalanced [] */
78: REG_ECOLLATE,  /* collation error - not relevant */
79: REG_ECTYPE,    /* bad class */
80: REG_EESCAPE,   /* bad escape sequence */
81: REG_EMPTY,     /* empty expression */
82: REG_EPAREN,    /* unbalanced () */
83: REG_ERANGE,    /* bad range inside [] */
84: REG_ESIZE,     /* expression too big */
85: REG_ESPACE,    /* failed to get memory */
86: REG_ESUBREG,   /* bad back reference */
87: REG_INVARG,    /* bad argument */
88: REG_NOMATCH    /* match failed */
89: };
90:
91:
92: /* The structure representing a compiled regular expression. */
93:
94: typedef struct {
95:     void *re_pcre;
96:     size_t re_nsub;
97:     size_t re_eroffset;
98: } regex_t;
99:
100: /* The structure in which a captured offset is returned. */
101:
102: typedef int regoff_t;
103:
104: typedef struct {
105:     regoff_t rm_so;
106:     regoff_t rm_eo;
107: } regmatch_t;
108:
109: /* When an application links to a PCRE DLL in Windows, the symbols that are
110: imported have to be identified as such. When building PCRE, the appropriate
111: export settings are needed, and are set in pcreposix.c before including this
112: file. */
113:
114: #if defined(_WIN32) && !defined(PCRE_STATIC) && !defined(PCREPOSIX_EXP_DECL)
115: # define PCREPOSIX_EXP_DECL extern __declspec(dllimport)
116: # define PCREPOSIX_EXP_DEFN __declspec(dllimport)

```

```

117: #endif
118:
119: /* By default, we use the standard "extern" declarations. */
120:
121: #ifndef PCREPOSIX_EXP_DECL
122: #  ifdef __cplusplus
123: #    define PCREPOSIX_EXP_DECL extern "C"
124: #    define PCREPOSIX_EXP_DEFN extern "C"
125: #  else
126: #    define PCREPOSIX_EXP_DECL extern
127: #    define PCREPOSIX_EXP_DEFN extern
128: #  endif
129: #endif
130:
131: /* The functions */
132:
133: PCREPOSIX_EXP_DECL int regcomp(regex_t *, const char *, int);
134: PCREPOSIX_EXP_DECL int regexec(const regex_t *, const char *, size_t,
135:                                regmatch_t *, int);
136: PCREPOSIX_EXP_DECL size_t regerror(int, const regex_t *, char *, size_t);
137: PCREPOSIX_EXP_DECL void regfree(regex_t *);
138:
139: #ifdef __cplusplus
140: } /* extern "C" */
141: #endif
142:
143: #endif /* End of pcreposix.h */

```

File: sdm/VxWorks/libRegex/pcre_compile.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_compile(), along with
42: supporting internal functions that are not used by other modules. */
43:
44:
45: #ifdef HAVE_CONFIG_H
46: #include "config.h"
47: #endif
48:
49: #define NLBLOCK cd      /* Block containing newline information */
50: #define PSSTART start_pattern /* Field containing processed string start */
51: #define PSEND end_pattern /* Field containing processed string end */
52:
53: #include "pcre_internal.h"
54:
55:
56: /* When DEBUG is defined, we need the pcre_printint() function, which is also
57: used by pcretest. DEBUG is not defined when building a production library. */
58:
59: #ifdef DEBUG
60: #include "pcre_printint.src"
61: #endif
62:
63:
64: /* Macro for setting individual bits in class bitmaps. */
65:
66: #define SETBIT(a,b) a[b/8] |= (1 << (b%8))
67:
68: /* Maximum length value to check against when making sure that the integer that
69: holds the compiled pattern length does not overflow. We make it a bit less than
70: INT_MAX to allow for adding in group terminating bytes, so that we don't have
71: to check them every time. */
72:
73: #define OFLOW_MAX (INT_MAX - 20)
74:
75:
76: /**

```

```

77: *    Code parameters and static tables    *
78: *****/
79:
80: /* This value specifies the size of stack workspace that is used during the
81: first pre-compile phase that determines how much memory is required. The regex
82: is partly compiled into this space, but the compiled parts are discarded as
83: soon as they can be, so that hopefully there will never be an overrun. The code
84: does, however, check for an overrun. The largest amount I've seen used is 218,
85: so this number is very generous.
86:
87: The same workspace is used during the second, actual compile phase for
88: remembering forward references to groups so that they can be filled in at the
89: end. Each entry in this list occupies LINK_SIZE bytes, so even when LINK_SIZE
90: is 4 there is plenty of room. */
91:
92: #define COMPILE_WORK_SIZE (4096)
93:
94:
95: /* Table for handling escaped characters in the range '0'-'z'. Positive returns
96: are simple data values; negative values are for special things like \d and so
97: on. Zero means further processing is needed (for things like \x), or the escape
98: is invalid. */
99:
100: #ifndef EBCDIC /* This is the "normal" table for ASCII systems */
101: static const short int escapes[] = {
102:     0,  0,  0,  0,  0,  0,  0,  0, /* 0 - 7 */
103:     0,  0,  ':', ':', '<', '=', '>', '?', /* 8 - ? */
104:     '@', -ESC_A, -ESC_B, -ESC_C, -ESC_D, -ESC_E,  0, -ESC_G, /* @ - G */
105:     -ESC_H,  0,  0, -ESC_K,  0,  0,  0,  0, /* H - O */
106:     -ESC_P, -ESC_Q, -ESC_R, -ESC_S,  0,  0, -ESC_V, -ESC_W, /* P - W */
107:     -ESC_X,  0, -ESC_Z, '[', '\\', ']', '^', '_', /* X - _ */
108:     ``, 7, -ESC_b,  0, -ESC_d, ESC_e, ESC_f,  0, /* ` - g */
109:     -ESC_h,  0,  0, -ESC_k,  0,  0, ESC_n,  0, /* h - o */
110:     -ESC_p,  0, ESC_r, -ESC_s, ESC_tee,  0, -ESC_v, -ESC_w, /* p - w */
111:     0,  0, -ESC_z, /* x - z */
112: };
113:
114: #else /* This is the "abnormal" table for EBCDIC systems */
115: static const short int escapes[] = {
116: /* 48 */ 0,  0,  0,  ':', '<', '(', '+', '|',
117: /* 50 */ '&',  0,  0,  0,  0,  0,  0,

```

```

118: /* 58 */ 0, 0, '!', '$', '*', ')', ';', '~',
119: /* 60 */ '-', '/', 0, 0, 0, 0, 0, 0,
120: /* 68 */ 0, 0, '|', ':', '%', '_', '>', '?',
121: /* 70 */ 0, 0, 0, 0, 0, 0, 0, 0,
122: /* 78 */ 0, '\', ':', '#', '@', '\', '=', '"',
123: /* 80 */ 0, 7, -ESC_b, 0, -ESC_d, ESC_e, ESC_f, 0,
124: /* 88 */ -ESC_h, 0, 0, '{', 0, 0, 0, 0,
125: /* 90 */ 0, 0, -ESC_k, 'l', 0, ESC_n, 0, -ESC_p,
126: /* 98 */ 0, ESC_r, 0, '}', 0, 0, 0, 0,
127: /* A0 */ 0, '~', -ESC_s, ESC_tee, 0, -ESC_v, -ESC_w, 0,
128: /* A8 */ 0, -ESC_z, 0, 0, 0, 0, '[', 0, 0,
129: /* B0 */ 0, 0, 0, 0, 0, 0, 0, 0,
130: /* B8 */ 0, 0, 0, 0, 0, ']', '=', '-',
131: /* C0 */ '{', -ESC_A, -ESC_B, -ESC_C, -ESC_D, -ESC_E, 0, -ESC_G,
132: /* C8 */ -ESC_H, 0, 0, 0, 0, 0, 0, 0,
133: /* D0 */ '}', 0, -ESC_K, 0, 0, 0, 0, -ESC_P,
134: /* D8 */ -ESC_Q, -ESC_R, 0, 0, 0, 0, 0, 0,
135: /* E0 */ '\\', 0, -ESC_S, 0, 0, -ESC_V, -ESC_W, -ESC_X,
136: /* E8 */ 0, -ESC_Z, 0, 0, 0, 0, 0, 0,
137: /* F0 */ 0, 0, 0, 0, 0, 0, 0, 0,
138: /* F8 */ 0, 0, 0, 0, 0, 0, 0, 0
139: };
140: #endif
141:
142:
143: /* Table of special "verbs" like (*PRUNE). This is a short table, so it is
144: searched linearly. Put all the names into a single string, in order to reduce
145: the number of relocations when a shared library is dynamically linked. */
146:
147: typedef struct verbitem {
148: int len;
149: int op;
150: } verbitem;
151:
152: static const char verbnames[] =
153: "ACCEPT \0"
154: "COMMIT \0"
155: "F \0"
156: "FAIL \0"
157: "PRUNE \0"
158: "SKIP \0"

```

```

159: "THEN";
160:
161: static const verbitem verbs[] = {
162: { 6, OP_ACCEPT },
163: { 6, OP_COMMIT },
164: { 1, OP_FAIL },
165: { 4, OP_FAIL },
166: { 5, OP_PRUNE },
167: { 4, OP_SKIP },
168: { 4, OP_THEN }
169: };
170:
171: static const int verbcount = sizeof(verbs)/sizeof(verbitem);
172:
173:
174: /* Tables of names of POSIX character classes and their lengths. The names are
175: now all in a single string, to reduce the number of relocations when a shared
176: library is dynamically loaded. The list of lengths is terminated by a zero
177: length entry. The first three must be alpha, lower, upper, as this is assumed
178: for handling case independence. */
179:
180: static const char posix_names[] =
181: "alpha \0" "lower \0" "upper \0" "alnum \0" "ascii \0" "blank \0"
182: "cntrl \0" "digit \0" "graph \0" "print \0" "punct \0" "space \0"
183: "word \0" "xdigit";
184:
185: static const uschar posix_name_lengths[] = {
186: 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 6, 0 };
187:
188: /* Table of class bit maps for each POSIX class. Each class is formed from a
189: base map, with an optional addition or removal of another map. Then, for some
190: classes, there is some additional tweaking: for [:blank:] the vertical space
191: characters are removed, and for [:alpha:] and [:alnum:] the underscore
192: character is removed. The triples in the table consist of the base map offset,
193: second map offset or -1 if no second map, and a non-negative value for map
194: addition or a negative value for map subtraction (if there are two maps). The
195: absolute value of the third field has these meanings: 0 => no tweaking, 1 =>
196: remove vertical space characters, 2 => remove underscore. */
197:
198: static const int posix_class_maps[] = {
199: cbit_word, cbit_digit, -2,          /* alpha */

```

```

200: cbit_lower, -1,    0,      /* lower */
201: cbit_upper, -1,    0,      /* upper */
202: cbit_word,  -1,    2,      /* alnum - word without underscore */
203: cbit_print, cbit_cntrl, 0,    /* ascii */
204: cbit_space, -1,    1,      /* blank - a GNU extension */
205: cbit_cntrl, -1,    0,      /* cntrl */
206: cbit_digit, -1,    0,      /* digit */
207: cbit_graph, -1,    0,      /* graph */
208: cbit_print, -1,    0,      /* print */
209: cbit_punct, -1,    0,      /* punct */
210: cbit_space, -1,    0,      /* space */
211: cbit_word,  -1,    0,      /* word - a Perl extension */
212: cbit_xdigit,-1,    0       /* xdigit */
213: };
214:
215:
216: #define STRING(a) #a
217: #define XSTRING(s) STRING(s)
218:
219: /* The texts of compile-time error messages. These are "char *" because they
220: are passed to the outside world. Do not ever re-use any error number, because
221: they are documented. Always add a new error instead. Messages marked DEAD below
222: are no longer used. This used to be a table of strings, but in order to reduce
223: the number of relocations needed when a shared library is loaded dynamically,
224: it is now one long string. We cannot use a table of offsets, because the
225: lengths of inserts such as XSTRING(MAX_NAME_SIZE) are not known. Instead, we
226: simply count through to the one we want - this isn't a performance issue
227: because these strings are used only when there is a compilation error. */
228:
229: static const char error_texts[] =
230: "no error \0"
231: "\\ at end of pattern \0"
232: "\\c at end of pattern \0"
233: "unrecognized character follows \\ \0"
234: "numbers out of order in {} quantifier \0"
235: /* 5 */
236: "number too big in {} quantifier \0"
237: "missing terminating ] for character class \0"
238: "invalid escape sequence in character class \0"
239: "range out of order in character class \0"
240: "nothing to repeat \0"

```


241: /* 10 */
 242: "operand of unlimited repeat could match the empty string \0" /** DEAD **/
 243: "internal error: unexpected repeat \0"
 244: "unrecognized character after (? or (?- \0"
 245: "POSIX named classes are supported only within a class \0"
 246: "missing) \0"
 247: /* 15 */
 248: "reference to non-existent subpattern \0"
 249: "erroffset passed as NULL \0"
 250: "unknown option bit(s) set \0"
 251: "missing) after comment \0"
 252: "parentheses nested too deeply \0" /** DEAD **/
 253: /* 20 */
 254: "regular expression is too large \0"
 255: "failed to get memory \0"
 256: "unmatched parentheses \0"
 257: "internal error: code overflow \0"
 258: "unrecognized character after (?< \0"
 259: /* 25 */
 260: "lookbehind assertion is not fixed length \0"
 261: "malformed number or name after (? \0"
 262: "conditional group contains more than two branches \0"
 263: "assertion expected after (? \0"
 264: "(?R or (?[+-]digits must be followed by) \0"
 265: /* 30 */
 266: "unknown POSIX class name \0"
 267: "POSIX collating elements are not supported \0"
 268: "this version of PCRE is not compiled with PCRE_UTF8 support \0"
 269: "spare error \0" /** DEAD **/
 270: "character value in \x{...} sequence is too large \0"
 271: /* 35 */
 272: "invalid condition (?0) \0"
 273: " \C not allowed in lookbehind assertion \0"
 274: "PCRE does not support \L, \l, \N, \U, or \u \0"
 275: "number after (?C is > 255 \0"
 276: "closing) for (?C expected \0"
 277: /* 40 */
 278: "recursive call could loop indefinitely \0"
 279: "unrecognized character after (?P \0"
 280: "syntax error in subpattern name (missing terminator) \0"
 281: "two named subpatterns have the same name \0"

282: "invalid UTF-8 string \0"
 283: /* 45 */
 284: "support for \\P, \\p, and \\X has not been compiled \0"
 285: "malformed \\P or \\p sequence \0"
 286: "unknown property name after \\P or \\p \0"
 287: "subpattern name is too long (maximum " XSTRING(MAX_NAME_SIZE) " characters) \0"
 288: "too many named subpatterns (maximum " XSTRING(MAX_NAME_COUNT) ") \0"
 289: /* 50 */
 290: "repeated subpattern is too long \0" /** DEAD **/
 291: "octal value is greater than \\377 (not in UTF-8 mode) \0"
 292: "internal error: overran compiling workspace \0"
 293: "internal error: previously-checked referenced subpattern not found \0"
 294: "DEFINE group contains more than one branch \0"
 295: /* 55 */
 296: "repeating a DEFINE group is not allowed \0"
 297: "inconsistent NEWLINE options \0"
 298: " \\g is not followed by a braced, angle-bracketed, or quoted name/number or by a plain number \0"
 299: "a numbered reference must not be zero \0"
 300: "(*VERB) with an argument is not supported \0"
 301: /* 60 */
 302: "(*VERB) not recognized \0"
 303: "number is too big \0"
 304: "subpattern name expected \0"
 305: "digit expected after (?+ \0"
 306: "]" is an invalid data character in JavaScript compatibility mode";
 307:
 308:
 309: /* Table to identify digits and hex digits. This is used when compiling
 310: patterns. Note that the tables in chartables are dependent on the locale, and
 311: may mark arbitrary characters as digits - but the PCRE compiling code expects
 312: to handle only 0-9, a-z, and A-Z as digits when compiling. That is why we have
 313: a private table here. It costs 256 bytes, but it is a lot faster than doing
 314: character value tests (at least in some simple cases I timed), and in some
 315: applications one wants PCRE to compile efficiently as well as match
 316: efficiently.
 317:
 318: For convenience, we use the same bit definitions as in chartables:
 319:
 320: 0x04 decimal digit
 321: 0x08 hexadecimal digit

```

322:
323: Then we can use ctype_digit and ctype_xdigit in the code. */
324:
325: #ifndef EBCDIC /* This is the "normal" case, for ASCII systems */
326: static const unsigned char digitab[] =
327: {
328: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 0- 7 */
329: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 8- 15 */
330: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 16- 23 */
331: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 24- 31 */
332: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /*  - ' */
333: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* ( - / */
334: 0x0c,0x0c,0x0c,0x0c,0x0c,0x0c,0x0c,0x0c, /* 0 - 7 */
335: 0x0c,0x0c,0x00,0x00,0x00,0x00,0x00,0x00, /* 8 - ? */
336: 0x00,0x08,0x08,0x08,0x08,0x08,0x08,0x00, /* @ - G */
337: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* H - O */
338: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* P - W */
339: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* X - _ */
340: 0x00,0x08,0x08,0x08,0x08,0x08,0x08,0x00, /* ` - g */
341: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* h - o */
342: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* p - w */
343: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* x -127 */
344: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 128-135 */
345: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 136-143 */
346: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 144-151 */
347: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 152-159 */
348: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 160-167 */
349: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 168-175 */
350: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 176-183 */
351: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 184-191 */
352: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 192-199 */
353: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 200-207 */
354: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 208-215 */
355: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 216-223 */
356: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 224-231 */
357: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 232-239 */
358: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 240-247 */
359: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}; /* 248-255 */
360:
361: #else /* This is the "abnormal" case, for EBCDIC systems */
362: static const unsigned char digitab[] =

```

```

363: {
364: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 0- 7 0 */
365: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 8- 15 */
366: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 16- 23 10 */
367: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 24- 31 */
368: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 32- 39 20 */
369: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 40- 47 */
370: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 48- 55 30 */
371: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 56- 63 */
372: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* - 71 40 */
373: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 72- | */
374: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* & - 87 50 */
375: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 88- 95 */
376: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* - -103 60 */
377: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 104- ? */
378: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 112-119 70 */
379: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 120- " */
380: 0x00,0x08,0x08,0x08,0x08,0x08,0x08,0x00, /* 128- g 80 */
381: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* h -143 */
382: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 144- p 90 */
383: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* q -159 */
384: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 160- x A0 */
385: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* y -175 */
386: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* ^ -183 B0 */
387: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 184-191 */
388: 0x00,0x08,0x08,0x08,0x08,0x08,0x08,0x00, /* { - G C0 */
389: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* H -207 */
390: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* } - P D0 */
391: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* Q -223 */
392: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* \ - X E0 */
393: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* Y -239 */
394: 0x0c,0x0c,0x0c,0x0c,0x0c,0x0c,0x0c,0x0c, /* 0 - 7 F0 */
395: 0x0c,0x0c,0x00,0x00,0x00,0x00,0x00,0x00}; /* 8 -255 */
396:
397: static const unsigned char ebcdic_chartab[] = { /* chartable partial dup */
398: 0x80,0x00,0x00,0x00,0x00,0x01,0x00,0x00, /* 0- 7 */
399: 0x00,0x00,0x00,0x00,0x01,0x01,0x00,0x00, /* 8- 15 */
400: 0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00, /* 16- 23 */
401: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 24- 31 */
402: 0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00, /* 32- 39 */
403: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 40- 47 */

```

```

404: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 48- 55 */
405: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 56- 63 */
406: 0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* - 71 */
407: 0x00,0x00,0x00,0x80,0x00,0x80,0x80,0x80, /* 72- | */
408: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* & - 87 */
409: 0x00,0x00,0x00,0x80,0x80,0x80,0x00,0x00, /* 88- 95 */
410: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* - -103 */
411: 0x00,0x00,0x00,0x00,0x00,0x10,0x00,0x80, /* 104- ? */
412: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 112-119 */
413: 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* 120- " */
414: 0x00,0x1a,0x1a,0x1a,0x1a,0x1a,0x1a,0x12, /* 128- g */
415: 0x12,0x12,0x00,0x00,0x00,0x00,0x00,0x00, /* h -143 */
416: 0x00,0x12,0x12,0x12,0x12,0x12,0x12,0x12, /* 144- p */
417: 0x12,0x12,0x00,0x00,0x00,0x00,0x00,0x00, /* q -159 */
418: 0x00,0x00,0x12,0x12,0x12,0x12,0x12,0x12, /* 160- x */
419: 0x12,0x12,0x00,0x00,0x00,0x00,0x00,0x00, /* y -175 */
420: 0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00, /* ^ -183 */
421: 0x00,0x00,0x80,0x00,0x00,0x00,0x00,0x00, /* 184-191 */
422: 0x80,0x1a,0x1a,0x1a,0x1a,0x1a,0x1a,0x12, /* { - G */
423: 0x12,0x12,0x00,0x00,0x00,0x00,0x00,0x00, /* H -207 */
424: 0x00,0x12,0x12,0x12,0x12,0x12,0x12,0x12, /* } - P */
425: 0x12,0x12,0x00,0x00,0x00,0x00,0x00,0x00, /* Q -223 */
426: 0x00,0x00,0x12,0x12,0x12,0x12,0x12,0x12, /* \ - X */
427: 0x12,0x12,0x00,0x00,0x00,0x00,0x00,0x00, /* Y -239 */
428: 0x1c,0x1c,0x1c,0x1c,0x1c,0x1c,0x1c,0x1c, /* 0 - 7 */
429: 0x1c,0x1c,0x00,0x00,0x00,0x00,0x00,0x00}; /* 8 -255 */
430: #endif
431:
432:
433: /* Definition to allow mutual recursion */
434:
435: static BOOL
436: compile_regex(int, int, uschar **, const uschar **, int *, BOOL, BOOL, int,
437: int *, int *, branch_chain *, compile_data *, int *);
438:
439:
440:
441: /*****
442: * Find an error text *
443: *****/
444:

```

```

445: /* The error texts are now all in one long string, to save on relocations. As
446: some of the text is of unknown length, we can't use a table of offsets.
447: Instead, just count through the strings. This is not a performance issue
448: because it happens only when there has been a compilation error.
449:
450: Argument:  the error number
451: Returns:   pointer to the error string
452: */
453:
454: static const char *
455: find_error_text(int n)
456: {
457:     const char *s = error_texts;
458:     for (; n > 0; n--) while (*s++ != 0) {};
459:     return s;
460: }
461:
462:
463: /*****
464:  *          Handle escapes          *
465:  *****/
466:
467: /* This function is called when a \ has been encountered. It either returns a
468: positive value for a simple escape such as \n, or a negative value which
469: encodes one of the more complicated things such as \d. A backreference to group
470: n is returned as -(ESC_REF + n); ESC_REF is the highest ESC_xxx macro. When
471: UTF-8 is enabled, a positive value greater than 255 may be returned. On entry,
472: ptr is pointing at the \. On exit, it is on the final character of the escape
473: sequence.
474:
475: Arguments:
476: ptrptr    points to the pattern position pointer
477: errorcodeptr points to the errorcode variable
478: bracount   number of previous extracting brackets
479: options    the options bits
480: isclass    TRUE if inside a character class
481:
482: Returns:   zero or positive => a data character
483:           negative => a special escape sequence
484:           on error, errorcodeptr is set
485: */

```

```

486:
487: static int
488: check_escape(const uschar **ptrptr, int *errorcodeptr, int bracount,
489: int options, BOOL isclass)
490: {
491:     BOOL utf8 = (options & PCRE_UTF8) != 0;
492:     const uschar *ptr = *ptrptr + 1;
493:     int c, i;
494:
495:     GETCHARINCTEST(c, ptr);      /* Get character value, increment pointer */
496:     ptr--;                      /* Set pointer back to the last byte */
497:
498:     /* If backslash is at the end of the pattern, it's an error. */
499:
500:     if (c == 0) *errorcodeptr = ERR1;
501:
502:     /* Non-alphanumerics are literals. For digits or letters, do an initial lookup
503:     in a table. A non-zero result is something that can be returned immediately.
504:     Otherwise further processing may be required. */
505:
506:     #ifndef EBCDIC /* ASCII coding */
507:     else if (c < '0' || c > 'z') {} /* Not alphanumeric */
508:     else if ((i = escapes[c - '0']) != 0) c = i;
509:
510:     #else /* EBCDIC coding */
511:     else if (c < 'a' || (ebcdic_chartab[c] & 0x0E) == 0) {} /* Not alphanumeric */
512:     else if ((i = escapes[c - 0x48]) != 0) c = i;
513:     #endif
514:
515:     /* Escapes that need further processing, or are illegal. */
516:
517:     else
518:     {
519:         const uschar *oldptr;
520:         BOOL braced, negated;
521:
522:         switch (c)
523:         {
524:             /* A number of Perl escapes are not handled by PCRE. We give an explicit
525:             error. */
526:

```

```

527: case 'l':
528: case 'L':
529: case 'N':
530: case 'u':
531: case 'U':
532: *errorcodeptr = ERR37;
533: break;
534:
535: /* \g must be followed by one of a number of specific things:
536:
537: (1) A number, either plain or braced. If positive, it is an absolute
538: backreference. If negative, it is a relative backreference. This is a Perl
539: 5.10 feature.
540:
541: (2) Perl 5.10 also supports \g{name} as a reference to a named group. This
542: is part of Perl's movement towards a unified syntax for back references. As
543: this is synonymous with \k{name}, we fudge it up by pretending it really
544: was \k.
545:
546: (3) For Oniguruma compatibility we also support \g followed by a name or a
547: number either in angle brackets or in single quotes. However, these are
548: (possibly recursive) subroutine calls, _not_ backreferences. Just return
549: the -ESC_g code (cf \k). */
550:
551: case 'g':
552: if (ptr[1] == '<' || ptr[1] == '\')
553: {
554: c = -ESC_g;
555: break;
556: }
557:
558: /* Handle the Perl-compatible cases */
559:
560: if (ptr[1] == '{')
561: {
562: const uschar *p;
563: for (p = ptr+2; *p != 0 && *p != '}'; p++)
564: if (*p != '-' && (digitab[*p] & ctype_digit) == 0) break;
565: if (*p != 0 && *p != '}')
566: {
567: c = -ESC_k;

```



```

568:     break;
569:     }
570:     braced = TRUE;
571:     ptr++;
572:     }
573: else braced = FALSE;
574:
575: if (ptr[1] == '-')
576:     {
577:     negated = TRUE;
578:     ptr++;
579:     }
580: else negated = FALSE;
581:
582: c = 0;
583: while ((digitab[ptr[1]] & ctype_digit) != 0)
584:     c = c * 10 + *(++ptr) - '0';
585:
586: if (c < 0) /* Integer overflow */
587:     {
588:     *errorcodeptr = ERR61;
589:     break;
590:     }
591:
592: if (braced && *(++ptr) != '}')
593:     {
594:     *errorcodeptr = ERR57;
595:     break;
596:     }
597:
598: if (c == 0)
599:     {
600:     *errorcodeptr = ERR58;
601:     break;
602:     }
603:
604: if (negated)
605:     {
606:     if (c > bracount)
607:         {
608:         *errorcodeptr = ERR15;

```

```

609:     break;
610: }
611: c = bracecount - (c - 1);
612: }
613:
614: c = -(ESC_REF + c);
615: break;
616:
617: /* The handling of escape sequences consisting of a string of digits
618: starting with one that is not zero is not straightforward. By experiment,
619: the way Perl works seems to be as follows:
620:
621: Outside a character class, the digits are read as a decimal number. If the
622: number is less than 10, or if there are that many previous extracting
623: left brackets, then it is a back reference. Otherwise, up to three octal
624: digits are read to form an escaped byte. Thus \123 is likely to be octal
625: 123 (cf \0123, which is octal 012 followed by the literal 3). If the octal
626: value is greater than 377, the least significant 8 bits are taken. Inside a
627: character class, \ followed by a digit is always an octal number. */
628:
629: case '1': case '2': case '3': case '4': case '5':
630: case '6': case '7': case '8': case '9':
631:
632: if (!isclass)
633: {
634:     oldptr = ptr;
635:     c -= '0';
636:     while ((digitab[ptr[1]] & ctype_digit) != 0)
637:         c = c * 10 + *(++ptr) - '0';
638:     if (c < 0) /* Integer overflow */
639:     {
640:         *errorcodeptr = ERR61;
641:         break;
642:     }
643:     if (c < 10 || c <= bracecount)
644:     {
645:         c = -(ESC_REF + c);
646:         break;
647:     }
648:     ptr = oldptr; /* Put the pointer back and fall through */
649: }

```

```

650:
651:  /* Handle an octal number following \. If the first digit is 8 or 9, Perl
652:  generates a binary zero byte and treats the digit as a following literal.
653:  Thus we have to pull back the pointer by one. */
654:
655:  if ((c = *ptr) >= '8')
656:  {
657:      ptr--;
658:      c = 0;
659:      break;
660:  }
661:
662:  /* \0 always starts an octal number, but we may drop through to here with a
663:  larger first octal digit. The original code used just to take the least
664:  significant 8 bits of octal numbers (I think this is what early Perls used
665:  to do). Nowadays we allow for larger numbers in UTF-8 mode, but no more
666:  than 3 octal digits. */
667:
668:  case '0':
669:      c -= '0';
670:      while(i++ < 2 && ptr[1] >= '0' && ptr[1] <= '7')
671:          c = c * 8 + *(++ptr) - '0';
672:      if (!utf8 && c > 255) *errorcodeptr = ERR51;
673:      break;
674:
675:  /* \x is complicated. \x{ddd} is a character number which can be greater
676:  than 0xff in utf8 mode, but only if the ddd are hex digits. If not, { is
677:  treated as a data character. */
678:
679:  case 'x':
680:      if (ptr[1] == '{')
681:      {
682:          const uschar *pt = ptr + 2;
683:          int count = 0;
684:
685:          c = 0;
686:          while ((digitab[*pt] & ctype_xdigit) != 0)
687:          {
688:              register int cc = *pt++;
689:              if (c == 0 && cc == '0') continue;  /* Leading zeroes */
690:              count++;

```

```

691:
692: #ifndef EBCDIC /* ASCII coding */
693:     if (cc >= 'a') cc -= 32;          /* Convert to upper case */
694:     c = (c << 4) + cc - ((cc < 'A')? '0' : ('A' - 10));
695: #else          /* EBCDIC coding */
696:     if (cc >= 'a' && cc <= 'z') cc += 64; /* Convert to upper case */
697:     c = (c << 4) + cc - ((cc >= '0')? '0' : ('A' - 10));
698: #endif
699:     }
700:
701:     if (*pt == '}')
702:     {
703:         if (c < 0 || count > (utf8? 8 : 2)) *errorcodeptr = ERR34;
704:         ptr = pt;
705:         break;
706:     }
707:
708:     /* If the sequence of hex digits does not end with '}', then we don't
709:     recognize this construct; fall through to the normal \x handling. */
710:     }
711:
712:     /* Read just a single-byte hex-defined char */
713:
714:     c = 0;
715:     while (i++ < 2 && (digitab[ptr[1]] & ctype_xdigit) != 0)
716:     {
717:         int cc;          /* Some compilers don't like ++ */
718:         cc = *(++ptr);    /* in initializers */
719: #ifndef EBCDIC /* ASCII coding */
720:         if (cc >= 'a') cc -= 32;          /* Convert to upper case */
721:         c = c * 16 + cc - ((cc < 'A')? '0' : ('A' - 10));
722: #else          /* EBCDIC coding */
723:         if (cc <= 'z') cc += 64;          /* Convert to upper case */
724:         c = c * 16 + cc - ((cc >= '0')? '0' : ('A' - 10));
725: #endif
726:     }
727:     break;
728:
729:     /* For \c, a following letter is upper-cased; then the 0x40 bit is flipped.
730:     This coding is ASCII-specific, but then the whole concept of \cx is
731:     ASCII-specific. (However, an EBCDIC equivalent has now been added.) */

```

```

732:
733: case 'c':
734:   c = *(++ptr);
735:   if (c == 0)
736:   {
737:     *errorcodeptr = ERR2;
738:     break;
739:   }
740:
741: #ifndef EBCDIC /* ASCII coding */
742:   if (c >= 'a' && c <= 'z') c -= 32;
743:   c ^= 0x40;
744: #else /* EBCDIC coding */
745:   if (c >= 'a' && c <= 'z') c += 64;
746:   c ^= 0xC0;
747: #endif
748:   break;
749:
750: /* PCRE_EXTRA enables extensions to Perl in the matter of escapes. Any
751: other alphanumeric following \ is an error if PCRE_EXTRA was set;
752: otherwise, for Perl compatibility, it is a literal. This code looks a bit
753: odd, but there used to be some cases other than the default, and there may
754: be again in future, so I haven't "optimized" it. */
755:
756: default:
757:   if ((options & PCRE_EXTRA) != 0) switch(c)
758:   {
759:     default:
760:       *errorcodeptr = ERR3;
761:       break;
762:   }
763:   break;
764: }
765: }
766:
767: *ptrptr = ptr;
768: return c;
769: }
770:
771:
772:

```

```

773: #ifdef SUPPORT_UCP
774: /*****
775:  *      Handle \P and \p      *
776:  *****/
777:
778: /* This function is called after \P or \p has been encountered, provided that
779: PCRE is compiled with support for Unicode properties. On entry, ptrptr is
780: pointing at the P or p. On exit, it is pointing at the final character of the
781: escape sequence.
782:
783: Argument:
784: ptrptr    points to the pattern position pointer
785: negptr     points to a boolean that is set TRUE for negation else FALSE
786: dptr       points to an int that is set to the detailed property value
787: errorcodeptr points to the error code variable
788:
789: Returns:    type value from ucp_type_table, or -1 for an invalid type
790: */
791:
792: static int
793: get_ucp(const uschar **ptrptr, BOOL *negptr, int *dptr, int *errorcodeptr)
794: {
795: int c, i, bot, top;
796: const uschar *ptr = *ptrptr;
797: char name[32];
798:
799: c = *(++ptr);
800: if (c == 0) goto ERROR_RETURN;
801:
802: *negptr = FALSE;
803:
804: /* \P or \p can be followed by a name in {}, optionally preceded by ^ for
805: negation. */
806:
807: if (c == '{')
808: {
809: if (ptr[1] == '^')
810: {
811: *negptr = TRUE;
812: ptr++;
813: }

```

```

814: for (i = 0; i < (int)sizeof(name) - 1; i++)
815: {
816:   c = *(++ptr);
817:   if (c == 0) goto ERROR_RETURN;
818:   if (c == '}') break;
819:   name[i] = c;
820: }
821: if (c != '}') goto ERROR_RETURN;
822: name[i] = 0;
823: }
824:
825: /* Otherwise there is just one following character */
826:
827: else
828: {
829:   name[0] = c;
830:   name[1] = 0;
831: }
832:
833: *ptrptr = ptr;
834:
835: /* Search for a recognized property name using binary chop */
836:
837: bot = 0;
838: top = _pcre_utt_size;
839:
840: while (bot < top)
841: {
842:   i = (bot + top) >> 1;
843:   c = strcmp(name, _pcre_utt_names + _pcre_utt[i].name_offset);
844:   if (c == 0)
845:   {
846:     *dptr = _pcre_utt[i].value;
847:     return _pcre_utt[i].type;
848:   }
849:   if (c > 0) bot = i + 1; else top = i;
850: }
851:
852: *errorcodeptr = ERR47;
853: *ptrptr = ptr;
854: return -1;

```

```

855:
856: ERROR_RETURN:
857: *errorcodeptr = ERR46;
858: *ptrptr = ptr;
859: return -1;
860: }
861: #endif
862:
863:
864:
865:
866: /*****
867:  *      Check for counted repeat      *
868:  *****/
869:
870: /* This function is called when a '{' is encountered in a place where it might
871: start a quantifier. It looks ahead to see if it really is a quantifier or not.
872: It is only a quantifier if it is one of the forms {ddd} {ddd,} or {ddd,ddd}
873: where the ddds are digits.
874:
875: Arguments:
876:  p      pointer to the first char after '{'
877:
878: Returns:  TRUE or FALSE
879: */
880:
881: static BOOL
882: is_counted_repeat(const uschar *p)
883: {
884: if ((digitab[*p++] & ctype_digit) == 0) return FALSE;
885: while ((digitab[*p] & ctype_digit) != 0) p++;
886: if (*p == '}') return TRUE;
887:
888: if (*p++ != ',') return FALSE;
889: if (*p == '}') return TRUE;
890:
891: if ((digitab[*p++] & ctype_digit) == 0) return FALSE;
892: while ((digitab[*p] & ctype_digit) != 0) p++;
893:
894: return (*p == '}');
895: }

```



```

896:
897:
898:
899: /*****
900: *      Read repeat counts      *
901: *****/
902:
903: /* Read an item of the form {n,m} and return the values. This is called only
904: after is_counted_repeat() has confirmed that a repeat-count quantifier exists,
905: so the syntax is guaranteed to be correct, but we need to check the values.
906:
907: Arguments:
908:  p          pointer to first char after '{'
909:  minp       pointer to int for min
910:  maxp       pointer to int for max
911:             returned as -1 if no max
912:  errorcodeptr points to error code variable
913:
914: Returns:    pointer to '}' on success;
915:             current ptr on error, with errorcodeptr set non-zero
916: */
917:
918: static const uschar *
919: read_repeat_counts(const uschar *p, int *minp, int *maxp, int *errorcodeptr)
920: {
921: int min = 0;
922: int max = -1;
923:
924: /* Read the minimum value and do a paranoid check: a negative value indicates
925: an integer overflow. */
926:
927: while ((digitab[*p] & ctype_digit) != 0) min = min * 10 + *p++ - '0';
928: if (min < 0 || min > 65535)
929: {
930: *errorcodeptr = ERR5;
931: return p;
932: }
933:
934: /* Read the maximum value if there is one, and again do a paranoid on its size.
935: Also, max must not be less than min. */
936:

```

```

937: if (*p == '}') max = min; else
938: {
939:   if (*(++p) != '}')
940:   {
941:     max = 0;
942:     while((digitab[*p] & ctype_digit) != 0) max = max * 10 + *p++ - '0';
943:     if (max < 0 || max > 65535)
944:     {
945:       *errorcodeptr = ERR5;
946:       return p;
947:     }
948:     if (max < min)
949:     {
950:       *errorcodeptr = ERR4;
951:       return p;
952:     }
953:   }
954: }
955:
956: /* Fill in the required variables, and pass back the pointer to the terminating
957:  *}'. */
958:
959: *minp = min;
960: *maxp = max;
961: return p;
962: }
963:
964:
965:
966: /*****
967:  *   Find forward referenced subpattern   *
968:  *****/
969:
970: /* This function scans along a pattern's text looking for capturing
971:  subpatterns, and counting them. If it finds a named pattern that matches the
972:  name it is given, it returns its number. Alternatively, if the name is NULL, it
973:  returns when it reaches a given numbered subpattern. This is used for forward
974:  references to subpatterns. We know that if (?P< is encountered, the name will
975:  be terminated by '>' because that is checked in the first pass.
976:
977: Arguments:

```

```

978: ptr      current position in the pattern
979: cd        compile background data
980: name      name to seek, or NULL if seeking a numbered subpattern
981: lorn      name length, or subpattern number if name is NULL
982: xmode     TRUE if we are in /x mode
983:
984: Returns:   the number of the named subpattern, or -1 if not found
985: */
986:
987: static int
988: find_parens(const uschar *ptr, compile_data *cd, const uschar *name, int lorn,
989:  BOOL xmode)
990: {
991: const uschar *thisname;
992: int count = cd->bracount;
993:
994: for (; *ptr != 0; ptr++)
995: {
996:  int term;
997:
998:  /* Skip over backslashed characters and also entire \Q... \E */
999:
1000:  if (*ptr == '\\')
1001:  {
1002:   if (*(++ptr) == 0) return -1;
1003:   if (*ptr == 'Q') for (;;)
1004:   {
1005:    while (*(++ptr) != 0 && *ptr != '\\') {};
1006:    if (*ptr == 0) return -1;
1007:    if (*(++ptr) == 'E') break;
1008:   }
1009:   continue;
1010:  }
1011:
1012:  /* Skip over character classes; this logic must be similar to the way they
1013:  1013:  are handled for real. If the first character is '^', skip it. Also, if the
1014:  1014:  first few characters (either before or after ^) are \Q \E or \E we skip them
1015:  1015:  too. This makes for compatibility with Perl. */
1016:
1017:  if (*ptr == '[')
1018:  {

```

```

1019:  BOOL negate_class = FALSE;
1020:  for (;;)
1021:  {
1022:      int c = *(++ptr);
1023:      if (c == '\\')
1024:      {
1025:          if (ptr[1] == 'E') ptr++;
1026:          else if (strncmp((const char *)ptr+1, "Q\\E", 3) == 0) ptr += 3;
1027:          else break;
1028:      }
1029:      else if (!negate_class && c == '^')
1030:          negate_class = TRUE;
1031:      else break;
1032:  }
1033:
1034:  /* If the next character is ']', it is a data character that must be
1035:     skipped, except in JavaScript compatibility mode. */
1036:
1037:  if (ptr[1] == ']' && (cd->external_options & PCRE_JAVASCRIPT_COMPAT) == 0)
1038:      ptr++;
1039:
1040:  while (*(++ptr) != ']')
1041:  {
1042:      if (*ptr == 0) return -1;
1043:      if (*ptr == '\\')
1044:      {
1045:          if (*(++ptr) == 0) return -1;
1046:          if (*ptr == 'Q') for (;;)
1047:          {
1048:              while (*(++ptr) != 0 && *ptr != '\\') {};
1049:              if (*ptr == 0) return -1;
1050:              if (*(++ptr) == 'E') break;
1051:          }
1052:          continue;
1053:      }
1054:  }
1055:  continue;
1056:  }
1057:
1058:  /* Skip comments in /x mode */
1059:

```

```

1060: if (xmode && *ptr == '#')
1061: {
1062: while (*(++ptr) != 0 && *ptr != '\n') {};
1063: if (*ptr == 0) return -1;
1064: continue;
1065: }
1066:
1067: /* An opening parens must now be a real metacharacter */
1068:
1069: if (*ptr != '(') continue;
1070: if (ptr[1] != '?' && ptr[1] != '*')
1071: {
1072: count++;
1073: if (name == NULL && count == lorn) return count;
1074: continue;
1075: }
1076:
1077: ptr += 2;
1078: if (*ptr == 'P') ptr++;          /* Allow optional P */
1079:
1080: /* We have to disambiguate (?<! and (?<= from (?<name> */
1081:
1082: if ((*ptr != '<' || ptr[1] == '!' || ptr[1] == '=') &&
1083:     *ptr != '\\')
1084: continue;
1085:
1086: count++;
1087:
1088: if (name == NULL && count == lorn) return count;
1089: term = *ptr++;
1090: if (term == '<') term = '>';
1091: thisname = ptr;
1092: while (*ptr != term) ptr++;
1093: if (name != NULL && lorn == ptr - thisname &&
1094:     strncmp((const char *)name, (const char *)thisname, lorn) == 0)
1095: return count;
1096: }
1097:
1098: return -1;
1099: }
1100:

```

```

1101:
1102:
1103: /*****
1104: *    Find first significant op code    *
1105: *****/
1106:
1107: /* This is called by several functions that scan a compiled expression looking
1108: for a fixed first character, or an anchoring op code etc. It skips over things
1109: that do not influence this. For some calls, a change of option is important.
1110: For some calls, it makes sense to skip negative forward and all backward
1111: assertions, and also the \b assertion; for others it does not.
1112:
1113: Arguments:
1114: code      pointer to the start of the group
1115: options   pointer to external options
1116: optbit    the option bit whose changing is significant, or
1117:           zero if none are
1118: skipassert TRUE if certain assertions are to be skipped
1119:
1120: Returns:   pointer to the first significant opcode
1121: */
1122:
1123: static const uschar*
1124: first_significant_code(const uschar *code, int *options, int optbit,
1125: BOOL skipassert)
1126: {
1127: for (;;)
1128: {
1129: switch ((int)*code)
1130: {
1131: case OP_OPT:
1132: if (optbit > 0 && ((int)code[1] & optbit) != (*options & optbit))
1133: *options = (int)code[1];
1134: code += 2;
1135: break;
1136:
1137: case OP_ASSERT_NOT:
1138: case OP_ASSERTBACK:
1139: case OP_ASSERTBACK_NOT:
1140: if (!skipassert) return code;
1141: do code += GET(code, 1); while (*code == OP_ALT);

```

```

1142: code += _pcre_OP_lengths[*code];
1143: break;
1144:
1145: case OP_WORD_BOUNDARY:
1146: case OP_NOT_WORD_BOUNDARY:
1147: if (!skipassert) return code;
1148: /* Fall through */
1149:
1150: case OP_CALLOUT:
1151: case OP_CREF:
1152: case OP_RREF:
1153: case OP_DEF:
1154: code += _pcre_OP_lengths[*code];
1155: break;
1156:
1157: default:
1158: return code;
1159: }
1160: }
1161: /* Control never reaches here */
1162: }
1163:
1164:
1165:
1166:
1167: /*****
1168: *      Find the fixed length of a pattern      *
1169: *****/
1170:
1171: /* Scan a pattern and compute the fixed length of subject that will match it,
1172: if the length is fixed. This is needed for dealing with backward assertions.
1173: In UTF8 mode, the result is in characters rather than bytes.
1174:
1175: Arguments:
1176: code    points to the start of the pattern (the bracket)
1177: options the compiling options
1178:
1179: Returns: the fixed length, or -1 if there is no fixed length,
1180:          or -2 if \C was encountered
1181: */
1182:

```

```

1183: static int
1184: find_fixedlength(uschar *code, int options)
1185: {
1186: int length = -1;
1187:
1188: register int branchlength = 0;
1189: register uschar *cc = code + 1 + LINK_SIZE;
1190:
1191: /* Scan along the opcodes for this branch. If we get to the end of the
1192: branch, check the length against that of the other branches. */
1193:
1194: for (;;)
1195: {
1196: int d;
1197: register int op = *cc;
1198: switch (op)
1199: {
1200: case OP_CBRA:
1201: case OP_BRA:
1202: case OP_ONCE:
1203: case OP_COND:
1204: d = find_fixedlength(cc + ((op == OP_CBRA)? 2:0), options);
1205: if (d < 0) return d;
1206: branchlength += d;
1207: do cc += GET(cc, 1); while (*cc == OP_ALT);
1208: cc += 1 + LINK_SIZE;
1209: break;
1210:
1211: /* Reached end of a branch; if it's a ket it is the end of a nested
1212: call. If it's ALT it is an alternation in a nested call. If it is
1213: END it's the end of the outer call. All can be handled by the same code. */
1214:
1215: case OP_ALT:
1216: case OP_KET:
1217: case OP_KETRMAT:
1218: case OP_KETRMIN:
1219: case OP_END:
1220: if (length < 0) length = branchlength;
1221: else if (length != branchlength) return -1;
1222: if (*cc != OP_ALT) return length;
1223: cc += 1 + LINK_SIZE;

```



```

1224:  branchlength = 0;
1225:  break;
1226:
1227:  /* Skip over assertive subpatterns */
1228:
1229:  case OP_ASSERT:
1230:  case OP_ASSERT_NOT:
1231:  case OP_ASSERTBACK:
1232:  case OP_ASSERTBACK_NOT:
1233:  do cc += GET(cc, 1); while (*cc == OP_ALT);
1234:  /* Fall through */
1235:
1236:  /* Skip over things that don't match chars */
1237:
1238:  case OP_REVERSE:
1239:  case OP_CREF:
1240:  case OP_RREF:
1241:  case OP_DEF:
1242:  case OP_OPT:
1243:  case OP_CALLOUT:
1244:  case OP_SOD:
1245:  case OP_SOM:
1246:  case OP_EOD:
1247:  case OP_EODN:
1248:  case OP_CIRC:
1249:  case OP_DOLL:
1250:  case OP_NOT_WORD_BOUNDARY:
1251:  case OP_WORD_BOUNDARY:
1252:  cc += _pcre_OP_lengths[*cc];
1253:  break;
1254:
1255:  /* Handle literal characters */
1256:
1257:  case OP_CHAR:
1258:  case OP_CHARNC:
1259:  case OP_NOT:
1260:  branchlength++;
1261:  cc += 2;
1262: #ifdef SUPPORT_UTF8
1263:  if ((options & PCRE_UTF8) != 0)
1264:  {

```

```

1265:   while ((*cc & 0xc0) == 0x80) cc++;
1266:   }
1267: #endif
1268:   break;
1269:
1270:   /* Handle exact repetitions. The count is already in characters, but we
1271:   need to skip over a multibyte character in UTF8 mode. */
1272:
1273:   case OP_EXACT:
1274:     branchlength += GET2(cc,1);
1275:     cc += 4;
1276: #ifdef SUPPORT_UTF8
1277:   if ((options & PCRE_UTF8) != 0)
1278:     {
1279:       while ((*cc & 0x80) == 0x80) cc++;
1280:     }
1281: #endif
1282:   break;
1283:
1284:   case OP_TYPEEXACT:
1285:     branchlength += GET2(cc,1);
1286:     if (cc[3] == OP_PROP || cc[3] == OP_NOTPROP) cc += 2;
1287:     cc += 4;
1288:     break;
1289:
1290:   /* Handle single-char matchers */
1291:
1292:   case OP_PROP:
1293:   case OP_NOTPROP:
1294:     cc += 2;
1295:   /* Fall through */
1296:
1297:   case OP_NOT_DIGIT:
1298:   case OP_DIGIT:
1299:   case OP_NOT_WHITESPACE:
1300:   case OP_WHITESPACE:
1301:   case OP_NOT_WORDCHAR:
1302:   case OP_WORDCHAR:
1303:   case OP_ANY:
1304:   case OP_ALLANY:
1305:     branchlength++;

```

```

1306:  cc++;
1307:  break;
1308:
1309:  /* The single-byte matcher isn't allowed */
1310:
1311:  case OP_ANYBYTE:
1312:  return -2;
1313:
1314:  /* Check a class for variable quantification */
1315:
1316: #ifdef SUPPORT_UTF8
1317:  case OP_XCLASS:
1318:  cc += GET(cc, 1) - 33;
1319:  /* Fall through */
1320: #endif
1321:
1322:  case OP_CLASS:
1323:  case OP_NCLASS:
1324:  cc += 33;
1325:
1326:  switch (*cc)
1327:  {
1328:  case OP_CRSTAR:
1329:  case OP_CRMINSTAR:
1330:  case OP_CRQUERY:
1331:  case OP_CRMINQUERY:
1332:  return -1;
1333:
1334:  case OP_CRRANGE:
1335:  case OP_CRMINRANGE:
1336:  if (GET2(cc,1) != GET2(cc,3)) return -1;
1337:  branchlength += GET2(cc,1);
1338:  cc += 5;
1339:  break;
1340:
1341:  default:
1342:  branchlength++;
1343:  }
1344:  break;
1345:
1346:  /* Anything else is variable length */

```

```

1347:
1348:  default:
1349:  return -1;
1350:  }
1351: }
1352: /* Control never gets here */
1353: }
1354:
1355:
1356:
1357:
1358: /*****
1359: *   Scan compiled regex for numbered bracket   *
1360: *****/
1361:
1362: /* This little function scans through a compiled pattern until it finds a
1363: capturing bracket with the given number.
1364:
1365: Arguments:
1366:  code    points to start of expression
1367:  utf8    TRUE in UTF-8 mode
1368:  number  the required bracket number
1369:
1370: Returns:  pointer to the opcode for the bracket, or NULL if not found
1371: */
1372:
1373: static const uschar *
1374: find_bracket(const uschar *code, BOOL utf8, int number)
1375: {
1376: for (;;)
1377: {
1378:  register int c = *code;
1379:  if (c == OP_END) return NULL;
1380:
1381:  /* XCLASS is used for classes that cannot be represented just by a bit
1382:  map. This includes negated single high-valued characters. The length in
1383:  the table is zero; the actual length is stored in the compiled code. */
1384:
1385:  if (c == OP_XCLASS) code += GET(code, 1);
1386:
1387:  /* Handle capturing bracket */

```

```

1388:
1389: else if (c == OP_CBRA)
1390: {
1391:     int n = GET2(code, 1+LINK_SIZE);
1392:     if (n == number) return (uschar *)code;
1393:     code += _pcre_OP_lengths[c];
1394: }
1395:
1396: /* Otherwise, we can get the item's length from the table, except that for
1397: repeated character types, we have to test for \p and \P, which have an extra
1398: two bytes of parameters. */
1399:
1400: else
1401: {
1402:     switch(c)
1403:     {
1404:     case OP_TYPESTAR:
1405:     case OP_TYPEMINSTAR:
1406:     case OP_TYPEPLUS:
1407:     case OP_TYPEMINPLUS:
1408:     case OP_TYPEQUERY:
1409:     case OP_TYPEMINQUERY:
1410:     case OP_TYPEPOSSTAR:
1411:     case OP_TYPEPOSPLUS:
1412:     case OP_TYPEPOSQUERY:
1413:     if (code[1] == OP_PROP || code[1] == OP_NOTPROP) code += 2;
1414:     break;
1415:
1416:     case OP_TYPEUPTO:
1417:     case OP_TYPEMINUPTO:
1418:     case OP_TYPEEXACT:
1419:     case OP_TYPEPOSUPTO:
1420:     if (code[3] == OP_PROP || code[3] == OP_NOTPROP) code += 2;
1421:     break;
1422:     }
1423:
1424:     /* Add in the fixed length from the table */
1425:
1426:     code += _pcre_OP_lengths[c];
1427:
1428:     /* In UTF-8 mode, opcodes that are followed by a character may be followed by

```

```

1429: a multi-byte character. The length in the table is a minimum, so we have to
1430: arrange to skip the extra bytes. */
1431:
1432: #ifdef SUPPORT_UTF8
1433:     if (utf8) switch(c)
1434:     {
1435:         case OP_CHAR:
1436:         case OP_CHARNC:
1437:         case OP_EXACT:
1438:         case OP_UPTO:
1439:         case OP_MINUPTO:
1440:         case OP_POSUPTO:
1441:         case OP_STAR:
1442:         case OP_MINSTAR:
1443:         case OP_POSSTAR:
1444:         case OP_PLUS:
1445:         case OP_MINPLUS:
1446:         case OP_POSPLUS:
1447:         case OP_QUERY:
1448:         case OP_MINQUERY:
1449:         case OP_POSQUERY:
1450:             if (code[-1] >= 0xc0) code += _pcre_utf8_table4[code[-1] & 0x3f];
1451:             break;
1452:     }
1453: #else
1454:     (void)(utf8); /* Keep compiler happy by referencing function argument */
1455: #endif
1456: }
1457: }
1458: }
1459:
1460:
1461:
1462: /*****
1463:  * Scan compiled regex for recursion reference *
1464:  *****/
1465:
1466: /* This little function scans through a compiled pattern until it finds an
1467: instance of OP_RECURSE.
1468:
1469: Arguments:

```

```

1470: code    points to start of expression
1471: utf8     TRUE in UTF-8 mode
1472:
1473: Returns:  pointer to the opcode for OP_RECURSE, or NULL if not found
1474: */
1475:
1476: static const uschar *
1477: find_recurse(const uschar *code, BOOL utf8)
1478: {
1479:     for (;;)
1480:     {
1481:         register int c = *code;
1482:         if (c == OP_END) return NULL;
1483:         if (c == OP_RECURSE) return code;
1484:
1485:         /* XCLASS is used for classes that cannot be represented just by a bit
1486:         map. This includes negated single high-valued characters. The length in
1487:         the table is zero; the actual length is stored in the compiled code. */
1488:
1489:         if (c == OP_XCLASS) code += GET(code, 1);
1490:
1491:         /* Otherwise, we can get the item's length from the table, except that for
1492:         repeated character types, we have to test for \p and \P, which have an extra
1493:         two bytes of parameters. */
1494:
1495:         else
1496:         {
1497:             switch(c)
1498:             {
1499:                 case OP_TYPESTAR:
1500:                 case OP_TYPEMINSTAR:
1501:                 case OP_TYPEPLUS:
1502:                 case OP_TYPEMINPLUS:
1503:                 case OP_TYPEQUERY:
1504:                 case OP_TYPEMINQUERY:
1505:                 case OP_TYPEPOSSTAR:
1506:                 case OP_TYPEPOSPLUS:
1507:                 case OP_TYPEPOSQUERY:
1508:                 if (code[1] == OP_PROP || code[1] == OP_NOTPROP) code += 2;
1509:                 break;
1510:

```

```

1511: case OP_TYPEPOSUPTO:
1512: case OP_TYPEUPTO:
1513: case OP_TYPEMINUPTO:
1514: case OP_TYPEEXACT:
1515: if (code[3] == OP_PROP || code[3] == OP_NOTPROP) code += 2;
1516: break;
1517: }
1518:
1519: /* Add in the fixed length from the table */
1520:
1521: code += _pcre_OP_lengths[c];
1522:
1523: /* In UTF-8 mode, opcodes that are followed by a character may be followed
1524: by a multi-byte character. The length in the table is a minimum, so we have
1525: to arrange to skip the extra bytes. */
1526:
1527: #ifdef SUPPORT_UTF8
1528: if (utf8) switch(c)
1529: {
1530: case OP_CHAR:
1531: case OP_CHARNC:
1532: case OP_EXACT:
1533: case OP_UPTO:
1534: case OP_MINUPTO:
1535: case OP_POSUPTO:
1536: case OP_STAR:
1537: case OP_MINSTAR:
1538: case OP_POSSTAR:
1539: case OP_PLUS:
1540: case OP_MINPLUS:
1541: case OP_POSPLUS:
1542: case OP_QUERY:
1543: case OP_MINQUERY:
1544: case OP_POSQUERY:
1545: if (code[-1] >= 0xc0) code += _pcre_utf8_table4[code[-1] & 0x3f];
1546: break;
1547: }
1548: #else
1549: (void)(utf8); /* Keep compiler happy by referencing function argument */
1550: #endif
1551: }

```



```

1552: }
1553: }
1554:
1555:
1556:
1557: /*****
1558: *   Scan compiled branch for non-emptiness   *
1559: *****/
1560:
1561: /* This function scans through a branch of a compiled pattern to see whether it
1562: can match the empty string or not. It is called from could_be_empty()
1563: below and from compile_branch() when checking for an unlimited repeat of a
1564: group that can match nothing. Note that first_significant_code() skips over
1565: backward and negative forward assertions when its final argument is TRUE. If we
1566: hit an unclosed bracket, we return "empty" - this means we've struck an inner
1567: bracket whose current branch will already have been scanned.
1568:
1569: Arguments:
1570: code      points to start of search
1571: endcode   points to where to stop
1572: utf8      TRUE if in UTF8 mode
1573:
1574: Returns:   TRUE if what is matched could be empty
1575: */
1576:
1577: static BOOL
1578: could_be_empty_branch(const uschar *code, const uschar *endcode, BOOL utf8)
1579: {
1580: register int c;
1581: for (code = first_significant_code(code + _pcre_OP_lengths[*code], NULL, 0, TRUE);
1582:      code < endcode;
1583:      code = first_significant_code(code + _pcre_OP_lengths[c], NULL, 0, TRUE))
1584: {
1585: const uschar *ccode;
1586:
1587: c = *code;
1588:
1589: /* Skip over forward assertions; the other assertions are skipped by
1590: first_significant_code() with a TRUE final argument. */
1591:
1592: if (c == OP_ASSERT)

```

```

1593:  {
1594:  do code += GET(code, 1); while (*code == OP_ALT);
1595:  c = *code;
1596:  continue;
1597:  }
1598:
1599:  /* Groups with zero repeats can of course be empty; skip them. */
1600:
1601:  if (c == OP_BRAZERO || c == OP_BRAMINZERO || c == OP_SKIPZERO)
1602:  {
1603:  code += _pcre_OP_lengths[c];
1604:  do code += GET(code, 1); while (*code == OP_ALT);
1605:  c = *code;
1606:  continue;
1607:  }
1608:
1609:  /* For other groups, scan the branches. */
1610:
1611:  if (c == OP_BRA || c == OP_CBRA || c == OP_ONCE || c == OP_COND)
1612:  {
1613:  BOOL empty_branch;
1614:  if (GET(code, 1) == 0) return TRUE;  /* Hit unclosed bracket */
1615:
1616:  /* Scan a closed bracket */
1617:
1618:  empty_branch = FALSE;
1619:  do
1620:  {
1621:  if (!empty_branch && could_be_empty_branch(code, endcode, utf8))
1622:  empty_branch = TRUE;
1623:  code += GET(code, 1);
1624:  }
1625:  while (*code == OP_ALT);
1626:  if (!empty_branch) return FALSE;  /* All branches are non-empty */
1627:  c = *code;
1628:  continue;
1629:  }
1630:
1631:  /* Handle the other opcodes */
1632:
1633:  switch (c)

```

```

1634: {
1635:  /* Check for quantifiers after a class. XCLASS is used for classes that
1636:  cannot be represented just by a bit map. This includes negated single
1637:  high-valued characters. The length in _pcre_OP_lengths[] is zero; the
1638:  actual length is stored in the compiled code, so we must update "code"
1639:  here. */
1640:
1641: #ifdef SUPPORT_UTF8
1642:  case OP_XCLASS:
1643:    ccode = code += GET(code, 1);
1644:    goto CHECK_CLASS_REPEAT;
1645: #endif
1646:
1647:  case OP_CLASS:
1648:  case OP_NCLASS:
1649:    ccode = code + 33;
1650:
1651: #ifdef SUPPORT_UTF8
1652:  CHECK_CLASS_REPEAT:
1653: #endif
1654:
1655:  switch (*ccode)
1656:  {
1657:    case OP_CRSTAR:      /* These could be empty; continue */
1658:    case OP_CRMINSTAR:
1659:    case OP_CRQUERY:
1660:    case OP_CRMINQUERY:
1661:      break;
1662:
1663:    default:             /* Non-repeat => class must match */
1664:    case OP_CRPLUS:      /* These repeats aren't empty */
1665:    case OP_CRMINPLUS:
1666:      return FALSE;
1667:
1668:    case OP_CRRANGE:
1669:    case OP_CRMINRANGE:
1670:      if (GET2(ccode, 1) > 0) return FALSE; /* Minimum > 0 */
1671:      break;
1672:  }
1673:  break;
1674:

```

```

1675:  /* Opcodes that must match a character */
1676:
1677:  case OP_PROP:
1678:  case OP_NOTPROP:
1679:  case OP_EXTUNI:
1680:  case OP_NOT_DIGIT:
1681:  case OP_DIGIT:
1682:  case OP_NOT_WHITESPACE:
1683:  case OP_WHITESPACE:
1684:  case OP_NOT_WORDCHAR:
1685:  case OP_WORDCHAR:
1686:  case OP_ANY:
1687:  case OP_ALLANY:
1688:  case OP_ANYBYTE:
1689:  case OP_CHAR:
1690:  case OP_CHARNC:
1691:  case OP_NOT:
1692:  case OP_PLUS:
1693:  case OP_MINPLUS:
1694:  case OP_POSPLUS:
1695:  case OP_EXACT:
1696:  case OP_NOTPLUS:
1697:  case OP_NOTMINPLUS:
1698:  case OP_NOTPOSPLUS:
1699:  case OP_NOTEXACT:
1700:  case OP_TYPEPLUS:
1701:  case OP_TYPEMINPLUS:
1702:  case OP_TYPEPOSPLUS:
1703:  case OP_TYPEEXACT:
1704:  return FALSE;
1705:
1706:  /* These are going to continue, as they may be empty, but we have to
1707:  fudge the length for the \p and \P cases. */
1708:
1709:  case OP_TYPESTAR:
1710:  case OP_TYPEMINSTAR:
1711:  case OP_TYPEPOSSTAR:
1712:  case OP_TYPEQUERY:
1713:  case OP_TYPEMINQUERY:
1714:  case OP_TYPEPOSQUERY:
1715:  if (code[1] == OP_PROP || code[1] == OP_NOTPROP) code += 2;

```

```

1716: break;
1717:
1718: /* Same for these */
1719:
1720: case OP_TYPEUPTO:
1721: case OP_TYPEMINUPTO:
1722: case OP_TYPEPOSUPTO:
1723: if (code[3] == OP_PROP || code[3] == OP_NOTPROP) code += 2;
1724: break;
1725:
1726: /* End of branch */
1727:
1728: case OP_KET:
1729: case OP_KETRMAX:
1730: case OP_KETRMIN:
1731: case OP_ALT:
1732: return TRUE;
1733:
1734: /* In UTF-8 mode, STAR, MINSTAR, POSSTAR, QUERY, MINQUERY, POSQUERY,
UPTO,
1735: MINUPTO, and POSUPTO may be followed by a multibyte character */
1736:
1737: #ifdef SUPPORT_UTF8
1738: case OP_STAR:
1739: case OP_MINSTAR:
1740: case OP_POSSTAR:
1741: case OP_QUERY:
1742: case OP_MINQUERY:
1743: case OP_POSQUERY:
1744: case OP_UPTO:
1745: case OP_MINUPTO:
1746: case OP_POSUPTO:
1747: if (utf8) while ((code[2] & 0xc0) == 0x80) code++;
1748: break;
1749: #endif
1750: }
1751: }
1752:
1753: return TRUE;
1754: }
1755:

```

```

1756:
1757:
1758: /*****
1759: *   Scan compiled regex for non-emptiness      *
1760: *****/
1761:
1762: /* This function is called to check for left recursive calls. We want to check
1763: the current branch of the current pattern to see if it could match the empty
1764: string. If it could, we must look outwards for branches at other levels,
1765: stopping when we pass beyond the bracket which is the subject of the recursion.
1766:
1767: Arguments:
1768:  code      points to start of the recursion
1769:  endcode   points to where to stop (current RECURSE item)
1770:  bcptr     points to the chain of current (unclosed) branch starts
1771:  utf8      TRUE if in UTF-8 mode
1772:
1773: Returns:   TRUE if what is matched could be empty
1774: */
1775:
1776: static BOOL
1777: could_be_empty(const uschar *code, const uschar *endcode, branch_chain *bcptr,
1778: BOOL utf8)
1779: {
1780: while (bcptr != NULL && bcptr->current >= code)
1781: {
1782: if (!could_be_empty_branch(bcptr->current, endcode, utf8)) return FALSE;
1783: bcptr = bcptr->outer;
1784: }
1785: return TRUE;
1786: }
1787:
1788:
1789:
1790: /*****
1791: *           Check for POSIX class syntax          *
1792: *****/
1793:
1794: /* This function is called when the sequence "[" or "[". or "["= is
1795: encountered in a character class. It checks whether this is followed by a
1796: sequence of characters terminated by a matching "]" or ".]" or "=]". If we

```

1797: reach an unescaped ']' without the special preceding character, return FALSE.

1798:

1799: Originally, this function only recognized a sequence of letters between the

1800: terminators, but it seems that Perl recognizes any sequence of characters,

1801: though of course unknown POSIX names are subsequently rejected. Perl gives an

1802: "Unknown POSIX class" error for [:f\oo:] for example, where previously PCRE

1803: didn't consider this to be a POSIX class. Likewise for [:1234:].

1804:

1805: The problem in trying to be exactly like Perl is in the handling of escapes. We

1806: have to be sure that [abc[:x\]pqr] is **not** treated as containing a POSIX

1807: class, but [abc[:x\]pqr:] is (so that an error can be generated). The code

1808: below handles the special case of \], but does not try to do any other escape

1809: processing. This makes it different from Perl for cases such as [:l\lower:]

1810: where Perl recognizes it as the POSIX class "lower" but PCRE does not recognize

1811: "l\lower". This is a lesser evil than not diagnosing bad classes when Perl does,

1812: I think.

1813:

1814: Arguments:

1815: ptr pointer to the initial [

1816: endptr where to return the end pointer

1817:

1818: Returns: TRUE or FALSE

1819: */

1820:

1821: static BOOL

1822: check_posix_syntax(const uschar *ptr, const uschar **endptr)

1823: {

1824: int terminator; /* Don't combine these lines; the Solaris cc */

1825: terminator = *(++ptr); /* compiler warns about "non-constant" initializer. */

1826: for (++ptr; *ptr != 0; ptr++)

1827: {

1828: if (*ptr == '\\\' && ptr[1] == ']') ptr++; else

1829: {

1830: if (*ptr == ']') return FALSE;

1831: if (*ptr == terminator && ptr[1] == ']')

1832: {

1833: *endptr = ptr;

1834: return TRUE;

1835: }

1836: }

1837: }

```

1838: return FALSE;
1839: }
1840:
1841:
1842:
1843:
1844: /*****
1845:  *      Check POSIX class name      *
1846: *****/
1847:
1848: /* This function is called to check the name given in a POSIX-style class entry
1849: such as [:alnum:].
1850:
1851: Arguments:
1852: ptr      points to the first letter
1853: len      the length of the name
1854:
1855: Returns:  a value representing the name, or -1 if unknown
1856: */
1857:
1858: static int
1859: check_posix_name(const uschar *ptr, int len)
1860: {
1861: const char *pn = posix_names;
1862: register int yield = 0;
1863: while (posix_name_lengths[yield] != 0)
1864: {
1865: if (len == posix_name_lengths[yield] &&
1866:     strcmp((const char *)ptr, pn, len) == 0) return yield;
1867: pn += posix_name_lengths[yield] + 1;
1868: yield++;
1869: }
1870: return -1;
1871: }
1872:
1873:
1874: /*****
1875:  *      Adjust OP_RECURSE items in repeated group      *
1876: *****/
1877:
1878: /* OP_RECURSE items contain an offset from the start of the regex to the group

```


1879: that is referenced. This means that groups can be replicated for fixed
1880: repetition simply by copying (because the recursion is allowed to refer to
1881: earlier groups that are outside the current group). However, when a group is
1882: optional (i.e. the minimum quantifier is zero), OP_BRAZERO or OP_SKIPZERO is
1883: inserted before it, after it has been compiled. This means that any OP_RECURSE
1884: items within it that refer to the group itself or any contained groups have to
1885: have their offsets adjusted. That one of the jobs of this function. Before it
1886: is called, the partially compiled regex must be temporarily terminated with
1887: OP_END.

1888:

1889: This function has been extended with the possibility of forward references for
1890: recursions and subroutine calls. It must also check the list of such references
1891: for the group we are dealing with. If it finds that one of the recursions in
1892: the current group is on this list, it adjusts the offset in the list, not the
1893: value in the reference (which is a group number).

1894:

1895: Arguments:

1896: group points to the start of the group

1897: adjust the amount by which the group is to be moved

1898: utf8 TRUE in UTF-8 mode

1899: cd contains pointers to tables etc.

1900: save_hwm the hwm forward reference pointer at the start of the group

1901:

1902: Returns: nothing

1903: */

1904:

1905: static void

1906: adjust_recurse(uschar *group, int adjust, BOOL utf8, compile_data *cd,

1907: uschar *save_hwm)

1908: {

1909: uschar *ptr = group;

1910:

1911: while ((ptr = (uschar *)find_recurse(ptr, utf8)) != NULL)

1912: {

1913: int offset;

1914: uschar *hc;

1915:

1916: /* See if this recursion is on the forward reference list. If so, adjust the

1917: reference. */

1918:

1919: for (hc = save_hwm; hc < cd->hwm; hc += LINK_SIZE)

```

1920: {
1921:   offset = GET(hc, 0);
1922:   if (cd->start_code + offset == ptr + 1)
1923:   {
1924:     PUT(hc, 0, offset + adjust);
1925:     break;
1926:   }
1927: }
1928:
1929: /* Otherwise, adjust the recursion offset if it's after the start of this
1930: group. */
1931:
1932: if (hc >= cd->hwm)
1933: {
1934:   offset = GET(ptr, 1);
1935:   if (cd->start_code + offset >= group) PUT(ptr, 1, offset + adjust);
1936: }
1937:
1938: ptr += 1 + LINK_SIZE;
1939: }
1940: }
1941:
1942:
1943:
1944: /*****
1945: *      Insert an automatic callout point      *
1946: *****/
1947:
1948: /* This function is called when the PCRE_AUTO_CALLOUT option is set, to insert
1949: callout points before each pattern item.
1950:
1951: Arguments:
1952: code      current code pointer
1953: ptr       current pattern pointer
1954: cd        pointers to tables etc
1955:
1956: Returns:   new code pointer
1957: */
1958:
1959: static uschar *
1960: auto_callout(uschar *code, const uschar *ptr, compile_data *cd)

```

```

1961: {
1962: *code++ = OP_CALLOUT;
1963: *code++ = 255;
1964: PUT(code, 0, ptr - cd->start_pattern); /* Pattern offset */
1965: PUT(code, LINK_SIZE, 0); /* Default length */
1966: return code + 2*LINK_SIZE;
1967: }
1968:
1969:
1970:
1971: /*****
1972: *      Complete a callout item      *
1973: *****/
1974:
1975: /* A callout item contains the length of the next item in the pattern, which
1976: we can't fill in till after we have reached the relevant point. This is used
1977: for both automatic and manual callouts.
1978:
1979: Arguments:
1980: previous_callout  points to previous callout item
1981: ptr              current pattern pointer
1982: cd               pointers to tables etc
1983:
1984: Returns:         nothing
1985: */
1986:
1987: static void
1988: complete_callout(uschar *previous_callout, const uschar *ptr, compile_data *cd)
1989: {
1990: int length = ptr - cd->start_pattern - GET(previous_callout, 2);
1991: PUT(previous_callout, 2 + LINK_SIZE, length);
1992: }
1993:
1994:
1995:
1996: #ifdef SUPPORT_UCP
1997: /*****
1998: *      Get othercase range      *
1999: *****/
2000:
2001: /* This function is passed the start and end of a class range, in UTF-8 mode

```

2002: with UCP support. It searches up the characters, looking for internal ranges of
 2003: characters in the "other" case. Each call returns the next one, updating the
 2004: start address.
 2005:
 2006: Arguments:
 2007: cptr points to starting character value; updated
 2008: d end value
 2009: ocptr where to put start of othercase range
 2010: odptr where to put end of othercase range
 2011:
 2012: Yield: TRUE when range returned; FALSE when no more
 2013: */
 2014:
 2015: static BOOL
 2016: get_othercase_range(unsigned int *cptr, unsigned int d, unsigned int *ocptr,
 2017: unsigned int *odptr)
 2018: {
 2019: unsigned int c, othercase, next;
 2020:
 2021: for (c = *cptr; c <= d; c++)
 2022: { if ((othercase = UCD_OTHERCASE(c)) != c) break; }
 2023:
 2024: if (c > d) return FALSE;
 2025:
 2026: *ocptr = othercase;
 2027: next = othercase + 1;
 2028:
 2029: for (++c; c <= d; c++)
 2030: {
 2031: if (UCD_OTHERCASE(c) != next) break;
 2032: next++;
 2033: }
 2034:
 2035: *odptr = next - 1;
 2036: *cptr = c;
 2037:
 2038: return TRUE;
 2039: }
 2040: #endif /* SUPPORT_UCP */
 2041:
 2042:

```

2043:
2044: /*****
2045: *   Check if auto-possessifying is possible   *
2046: *****/
2047:
2048: /* This function is called for unlimited repeats of certain items, to see
2049: whether the next thing could possibly match the repeated item. If not, it makes
2050: sense to automatically possessify the repeated item.
2051:
2052: Arguments:
2053: op_code    the repeated op code
2054: this       data for this item, depends on the opcode
2055: utf8       TRUE in UTF-8 mode
2056: utf8_char  used for utf8 character bytes, NULL if not relevant
2057: ptr        next character in pattern
2058: options    options bits
2059: cd         contains pointers to tables etc.
2060:
2061: Returns:    TRUE if possessifying is wanted
2062: */
2063:
2064: static BOOL
2065: check_auto_possessive(int op_code, int item, BOOL utf8, uschar *utf8_char,
2066: const uschar *ptr, int options, compile_data *cd)
2067: {
2068: int next;
2069:
2070: /* Skip whitespace and comments in extended mode */
2071:
2072: if ((options & PCRE_EXTENDED) != 0)
2073: {
2074: for (;;)
2075: {
2076: while ((cd->ctypes[*ptr] & ctype_space) != 0) ptr++;
2077: if (*ptr == '#')
2078: {
2079: while (*(++ptr) != 0)
2080: if (IS_NEWLINE(ptr)) { ptr += cd->nllen; break; }
2081: }
2082: else break;
2083: }

```

```

2084: }
2085:
2086: /* If the next item is one that we can handle, get its value. A non-negative
2087: value is a character, a negative value is an escape value. */
2088:
2089: if (*ptr == '\\')
2090: {
2091:     int temperrorcode = 0;
2092:     next = check_escape(&ptr, &temperrorcode, cd->bracount, options, FALSE);
2093:     if (temperrorcode != 0) return FALSE;
2094:     ptr++; /* Point after the escape sequence */
2095: }
2096:
2097: else if ((cd->ctypes[*ptr] & ctype_meta) == 0)
2098: {
2099: #ifdef SUPPORT_UTF8
2100:     if (utf8) { GETCHARINC(next, ptr); } else
2101: #endif
2102:     next = *ptr++;
2103: }
2104:
2105: else return FALSE;
2106:
2107: /* Skip whitespace and comments in extended mode */
2108:
2109: if ((options & PCRE_EXTENDED) != 0)
2110: {
2111:     for (;;)
2112:     {
2113:         while ((cd->ctypes[*ptr] & ctype_space) != 0) ptr++;
2114:         if (*ptr == '#')
2115:         {
2116:             while (*(++ptr) != 0)
2117:                 if (IS_NEWLINE(ptr)) { ptr += cd->nllen; break; }
2118:         }
2119:         else break;
2120:     }
2121: }
2122:
2123: /* If the next thing is itself optional, we have to give up. */
2124:

```

```

2125: if (*ptr == '*' || *ptr == '?' || strcmp((char *)ptr, "{0,", 3) == 0)
2126:   return FALSE;
2127:
2128: /* Now compare the next item with the previous opcode. If the previous is a
2129: positive single character match, "item" either contains the character or, if
2130: "item" is greater than 127 in utf8 mode, the character's bytes are in
2131: utf8_char. */
2132:
2133:
2134: /* Handle cases when the next item is a character. */
2135:
2136: if (next >= 0) switch(op_code)
2137: {
2138:   case OP_CHAR:
2139: #ifdef SUPPORT_UTF8
2140:   if (utf8 && item > 127) { GETCHAR(item, utf8_char); }
2141: #else
2142:   (void)(utf8_char); /* Keep compiler happy by referencing function argument */
2143: #endif
2144:   return item != next;
2145:
2146:   /* For CHARNC (caseless character) we must check the other case. If we have
2147:   Unicode property support, we can use it to test the other case of
2148:   high-valued characters. */
2149:
2150:   case OP_CHARNC:
2151: #ifdef SUPPORT_UTF8
2152:   if (utf8 && item > 127) { GETCHAR(item, utf8_char); }
2153: #endif
2154:   if (item == next) return FALSE;
2155: #ifdef SUPPORT_UTF8
2156:   if (utf8)
2157:   {
2158:     unsigned int othercase;
2159:     if (next < 128) othercase = cd->fcc[next]; else
2160: #ifdef SUPPORT_UCP
2161:     othercase = UCD_OTHERCASE((unsigned int)next);
2162: #else
2163:     othercase = NOTACHAR;
2164: #endif
2165:     return (unsigned int)item != othercase;

```

```

2166: }
2167: else
2168: #endif /* SUPPORT_UTF8 */
2169: return (item != cd->fcc[next]); /* Non-UTF-8 mode */
2170:
2171: /* For OP_NOT, "item" must be a single-byte character. */
2172:
2173: case OP_NOT:
2174: if (item == next) return TRUE;
2175: if ((options & PCRE_CASELESS) == 0) return FALSE;
2176: #ifdef SUPPORT_UTF8
2177: if (utf8)
2178: {
2179: unsigned int othercase;
2180: if (next < 128) othercase = cd->fcc[next]; else
2181: #ifdef SUPPORT_UCP
2182: othercase = UCD_OTHERCASE(next);
2183: #else
2184: othercase = NOTACHAR;
2185: #endif
2186: return (unsigned int)item == othercase;
2187: }
2188: else
2189: #endif /* SUPPORT_UTF8 */
2190: return (item == cd->fcc[next]); /* Non-UTF-8 mode */
2191:
2192: case OP_DIGIT:
2193: return next > 127 || (cd->ctypes[next] & ctype_digit) == 0;
2194:
2195: case OP_NOT_DIGIT:
2196: return next <= 127 && (cd->ctypes[next] & ctype_digit) != 0;
2197:
2198: case OP_WHITESPACE:
2199: return next > 127 || (cd->ctypes[next] & ctype_space) == 0;
2200:
2201: case OP_NOT_WHITESPACE:
2202: return next <= 127 && (cd->ctypes[next] & ctype_space) != 0;
2203:
2204: case OP_WORDCHAR:
2205: return next > 127 || (cd->ctypes[next] & ctype_word) == 0;
2206:

```



```

2207: case OP_NOT_WORDCHAR:
2208: return next <= 127 && (cd->ctypes[next] & ctype_word) != 0;
2209:
2210: case OP_HSPACE:
2211: case OP_NOT_HSPACE:
2212: switch(next)
2213: {
2214:     case 0x09:
2215:     case 0x20:
2216:     case 0xa0:
2217:     case 0x1680:
2218:     case 0x180e:
2219:     case 0x2000:
2220:     case 0x2001:
2221:     case 0x2002:
2222:     case 0x2003:
2223:     case 0x2004:
2224:     case 0x2005:
2225:     case 0x2006:
2226:     case 0x2007:
2227:     case 0x2008:
2228:     case 0x2009:
2229:     case 0x200A:
2230:     case 0x202f:
2231:     case 0x205f:
2232:     case 0x3000:
2233: return op_code != OP_HSPACE;
2234: default:
2235: return op_code == OP_HSPACE;
2236: }
2237:
2238: case OP_VSPACE:
2239: case OP_NOT_VSPACE:
2240: switch(next)
2241: {
2242:     case 0x0a:
2243:     case 0x0b:
2244:     case 0x0c:
2245:     case 0x0d:
2246:     case 0x85:
2247:     case 0x2028:

```

```

2248: case 0x2029:
2249: return op_code != OP_VSPACE;
2250: default:
2251: return op_code == OP_VSPACE;
2252: }
2253:
2254: default:
2255: return FALSE;
2256: }
2257:
2258:
2259: /* Handle the case when the next item is \d, \s, etc. */
2260:
2261: switch(op_code)
2262: {
2263: case OP_CHAR:
2264: case OP_CHARNC:
2265: #ifdef SUPPORT_UTF8
2266: if (utf8 && item > 127) { GETCHAR(item, utf8_char); }
2267: #endif
2268: switch(-next)
2269: {
2270: case ESC_d:
2271: return item > 127 || (cd->ctypes[item] & ctype_digit) == 0;
2272:
2273: case ESC_D:
2274: return item <= 127 && (cd->ctypes[item] & ctype_digit) != 0;
2275:
2276: case ESC_s:
2277: return item > 127 || (cd->ctypes[item] & ctype_space) == 0;
2278:
2279: case ESC_S:
2280: return item <= 127 && (cd->ctypes[item] & ctype_space) != 0;
2281:
2282: case ESC_w:
2283: return item > 127 || (cd->ctypes[item] & ctype_word) == 0;
2284:
2285: case ESC_W:
2286: return item <= 127 && (cd->ctypes[item] & ctype_word) != 0;
2287:
2288: case ESC_h:

```

```

2289: case ESC_H:
2290: switch(item)
2291: {
2292: case 0x09:
2293: case 0x20:
2294: case 0xa0:
2295: case 0x1680:
2296: case 0x180e:
2297: case 0x2000:
2298: case 0x2001:
2299: case 0x2002:
2300: case 0x2003:
2301: case 0x2004:
2302: case 0x2005:
2303: case 0x2006:
2304: case 0x2007:
2305: case 0x2008:
2306: case 0x2009:
2307: case 0x200A:
2308: case 0x202f:
2309: case 0x205f:
2310: case 0x3000:
2311: return -next != ESC_h;
2312: default:
2313: return -next == ESC_h;
2314: }
2315:
2316: case ESC_v:
2317: case ESC_V:
2318: switch(item)
2319: {
2320: case 0x0a:
2321: case 0x0b:
2322: case 0x0c:
2323: case 0x0d:
2324: case 0x85:
2325: case 0x2028:
2326: case 0x2029:
2327: return -next != ESC_v;
2328: default:
2329: return -next == ESC_v;

```

```

2330:     }
2331:
2332:     default:
2333:         return FALSE;
2334:     }
2335:
2336:     case OP_DIGIT:
2337:         return next == -ESC_D || next == -ESC_s || next == -ESC_W ||
2338:             next == -ESC_h || next == -ESC_v;
2339:
2340:     case OP_NOT_DIGIT:
2341:         return next == -ESC_d;
2342:
2343:     case OP_WHITESPACE:
2344:         return next == -ESC_S || next == -ESC_d || next == -ESC_w;
2345:
2346:     case OP_NOT_WHITESPACE:
2347:         return next == -ESC_s || next == -ESC_h || next == -ESC_v;
2348:
2349:     case OP_HSPACE:
2350:         return next == -ESC_S || next == -ESC_H || next == -ESC_d || next == -ESC_w;
2351:
2352:     case OP_NOT_HSPACE:
2353:         return next == -ESC_h;
2354:
2355:     /* Can't have \S in here because VT matches \S (Perl anomaly) */
2356:     case OP_VSPACE:
2357:         return next == -ESC_V || next == -ESC_d || next == -ESC_w;
2358:
2359:     case OP_NOT_VSPACE:
2360:         return next == -ESC_v;
2361:
2362:     case OP_WORDCHAR:
2363:         return next == -ESC_W || next == -ESC_s || next == -ESC_h || next == -ESC_v;
2364:
2365:     case OP_NOT_WORDCHAR:
2366:         return next == -ESC_w || next == -ESC_d;
2367:
2368:     default:
2369:         return FALSE;
2370:     }

```

```

2371:
2372: /* Control does not reach here */
2373: }
2374:
2375:
2376:
2377: /*****
2378: *      Compile one branch      *
2379: *****/
2380:
2381: /* Scan the pattern, compiling it into the a vector. If the options are
2382: changed during the branch, the pointer is used to change the external options
2383: bits. This function is used during the pre-compile phase when we are trying
2384: to find out the amount of memory needed, as well as during the real compile
2385: phase. The value of lengthptr distinguishes the two phases.
2386:
2387: Arguments:
2388: optionsptr  pointer to the option bits
2389: codeptr     points to the pointer to the current code point
2390: ptrptr      points to the current pattern pointer
2391: errorcodeptr points to error code variable
2392: firstbyteptr set to initial literal character, or < 0 (REQ_UNSET, REQ_NONE)
2393: reqbyteptr   set to the last literal character required, else < 0
2394: bcptr        points to current branch chain
2395: cd           contains pointers to tables etc.
2396: lengthptr    NULL during the real compile phase
2397:              points to length accumulator during pre-compile phase
2398:
2399: Returns:      TRUE on success
2400:              FALSE, with *errorcodeptr set non-zero on error
2401: */
2402:
2403: static BOOL
2404: compile_branch(int *optionsptr, uschar **codeptr, const uschar **ptrptr,
2405: int *errorcodeptr, int *firstbyteptr, int *reqbyteptr, branch_chain *bcptr,
2406: compile_data *cd, int *lengthptr)
2407: {
2408: int repeat_type, op_type;
2409: int repeat_min = 0, repeat_max = 0; /* To please picky compilers */
2410: int bravalue = 0;
2411: int greedy_default, greedy_non_default;

```

```

2412: int firstbyte, reqbyte;
2413: int zeroreqbyte, zerofirstbyte;
2414: int req_caseopt, reqvary, tempreqvary;
2415: int options = *optionsptr;
2416: int after_manual_callout = 0;
2417: int length_prevgroup = 0;
2418: register int c;
2419: register uschar *code = *codeptr;
2420: uschar *last_code = code;
2421: uschar *orig_code = code;
2422: uschar *tempcode;
2423: BOOL inescq = FALSE;
2424: BOOL groupsetfirstbyte = FALSE;
2425: const uschar *ptr = *ptrptr;
2426: const uschar *tempptr;
2427: uschar *previous = NULL;
2428: uschar *previous_callout = NULL;
2429: uschar *save_hwm = NULL;
2430: uschar classbits[32];
2431:
2432: #ifdef SUPPORT_UTF8
2433: BOOL class_utf8;
2434: BOOL utf8 = (options & PCRE_UTF8) != 0;
2435: uschar *class_utf8data;
2436: uschar *class_utf8data_base;
2437: uschar utf8_char[6];
2438: #else
2439: BOOL utf8 = FALSE;
2440: uschar *utf8_char = NULL;
2441: #endif
2442:
2443: #ifdef DEBUG
2444: if (lengthptr != NULL) DPRINTF((">> start branch \n"));
2445: #endif
2446:
2447: /* Set up the default and non-default settings for greediness */
2448:
2449: greedy_default = ((options & PCRE_UNGREEDY) != 0);
2450: greedy_non_default = greedy_default ^ 1;
2451:
2452: /* Initialize no first byte, no required byte. REQ_UNSET means "no char

```

```

2453: matching encountered yet". It gets changed to REQ_NONE if we hit something that
2454: matches a non-fixed char first char; reqbyte just remains unset if we never
2455: find one.
2456:
2457: When we hit a repeat whose minimum is zero, we may have to adjust these values
2458: to take the zero repeat into account. This is implemented by setting them to
2459: zerofirstbyte and zeroreqbyte when such a repeat is encountered. The individual
2460: item types that can be repeated set these backoff variables appropriately. */
2461:
2462: firstbyte = reqbyte = zerofirstbyte = zeroreqbyte = REQ_UNSET;
2463:
2464: /* The variable req_caseopt contains either the REQ_CASELESS value or zero,
2465: according to the current setting of the caseless flag. REQ_CASELESS is a bit
2466: value > 255. It is added into the firstbyte or reqbyte variables to record the
2467: case status of the value. This is used only for ASCII characters. */
2468:
2469: req_caseopt = ((options & PCRE_CASELESS) != 0)? REQ_CASELESS : 0;
2470:
2471: /* Switch on next character until the end of the branch */
2472:
2473: for (;;) ptr++)
2474: {
2475:   BOOL negate_class;
2476:   BOOL should_flip_negation;
2477:   BOOL possessive_quantifier;
2478:   BOOL is_quantifier;
2479:   BOOL is_recurse;
2480:   BOOL reset_bracount;
2481:   int class_charcount;
2482:   int class_lastchar;
2483:   int newoptions;
2484:   int recno;
2485:   int refsing;
2486:   int skipbytes;
2487:   int subreqbyte;
2488:   int subfirstbyte;
2489:   int terminator;
2490:   int mclength;
2491:   uschar mcbuffer[8];
2492:
2493:   /* Get next byte in the pattern */

```

```

2494:
2495: c = *ptr;
2496:
2497: /* If we are in the pre-compile phase, accumulate the length used for the
2498: previous cycle of this loop. */
2499:
2500: if (lengthptr != NULL)
2501: {
2502: #ifdef DEBUG
2503:   if (code > cd->hwm) cd->hwm = code;          /* High water info */
2504: #endif
2505:   if (code > cd->start_workspace + COMPILE_WORK_SIZE) /* Check for overrun */
2506:   {
2507:     *errorcodeptr = ERR52;
2508:     goto FAILED;
2509:   }
2510:
2511:   /* There is at least one situation where code goes backwards: this is the
2512:   case of a zero quantifier after a class (e.g. [ab]{0}). At compile time,
2513:   the class is simply eliminated. However, it is created first, so we have to
2514:   allow memory for it. Therefore, don't ever reduce the length at this point.
2515:   */
2516:
2517:   if (code < last_code) code = last_code;
2518:
2519:   /* Paranoid check for integer overflow */
2520:
2521:   if (OFLOW_MAX - *lengthptr < code - last_code)
2522:   {
2523:     *errorcodeptr = ERR20;
2524:     goto FAILED;
2525:   }
2526:
2527:   *lengthptr += code - last_code;
2528:   DPRINTF(("length=%d added %d c=%c\n", *lengthptr, code - last_code, c));
2529:
2530:   /* If "previous" is set and it is not at the start of the work space, move
2531:   it back to there, in order to avoid filling up the work space. Otherwise,
2532:   if "previous" is NULL, reset the current code pointer to the start. */
2533:
2534:   if (previous != NULL)

```



```

2535:  {
2536:  if (previous > orig_code)
2537:  {
2538:      memmove(orig_code, previous, code - previous);
2539:      code -= previous - orig_code;
2540:      previous = orig_code;
2541:  }
2542:  }
2543:  else code = orig_code;
2544:
2545:  /* Remember where this code item starts so we can pick up the length
2546:  next time round. */
2547:
2548:  last_code = code;
2549:  }
2550:
2551:  /* In the real compile phase, just check the workspace used by the forward
2552:  reference list. */
2553:
2554:  else if (cd->hwm > cd->start_workspace + COMPILE_WORK_SIZE)
2555:  {
2556:      *errorcodeptr = ERR52;
2557:      goto FAILED;
2558:  }
2559:
2560:  /* If in \Q... \E, check for the end; if not, we have a literal */
2561:
2562:  if (inescq && c != 0)
2563:  {
2564:      if (c == '\\' && ptr[1] == 'E')
2565:      {
2566:          inescq = FALSE;
2567:          ptr++;
2568:          continue;
2569:      }
2570:      else
2571:      {
2572:          if (previous_callout != NULL)
2573:          {
2574:              if (lengthptr == NULL) /* Don't attempt in pre-compile phase */
2575:                  complete_callout(previous_callout, ptr, cd);

```

```

2576:     previous_callout = NULL;
2577: }
2578: if ((options & PCRE_AUTO_CALLOUT) != 0)
2579: {
2580:     previous_callout = code;
2581:     code = auto_callout(code, ptr, cd);
2582: }
2583: goto NORMAL_CHAR;
2584: }
2585: }
2586:
2587: /* Fill in length of a previous callout, except when the next thing is
2588: a quantifier. */
2589:
2590: is_quantifier = c == '*' || c == '+' || c == '?' ||
2591: (c == '{' && is_counted_repeat(ptr+1));
2592:
2593: if (!is_quantifier && previous_callout != NULL &&
2594:     after_manual_callout-- <= 0)
2595: {
2596:     if (lengthptr == NULL) /* Don't attempt in pre-compile phase */
2597:         complete_callout(previous_callout, ptr, cd);
2598:     previous_callout = NULL;
2599: }
2600:
2601: /* In extended mode, skip white space and comments */
2602:
2603: if ((options & PCRE_EXTENDED) != 0)
2604: {
2605:     if ((cd->ctypes[c] & ctype_space) != 0) continue;
2606:     if (c == '#')
2607:     {
2608:         while (*(++ptr) != 0)
2609:         {
2610:             if (IS_NEWLINE(ptr)) { ptr += cd->nllen - 1; break; }
2611:         }
2612:         if (*ptr != 0) continue;
2613:
2614:         /* Else fall through to handle end of string */
2615:         c = 0;
2616:     }

```

```

2617: }
2618:
2619: /* No auto callout for quantifiers. */
2620:
2621: if ((options & PCRE_AUTO_CALLOUT) != 0 && !is_quantifier)
2622: {
2623:     previous_callout = code;
2624:     code = auto_callout(code, ptr, cd);
2625: }
2626:
2627: switch(c)
2628: {
2629:
=====*/
2630:     case 0:          /* The branch terminates at string end */
2631:     case '|':        /* or | or ) */
2632:     case ')':
2633:         *firstbyteptr = firstbyte;
2634:         *reqbyteptr = reqbyte;
2635:         *codeptr = code;
2636:         *ptrptr = ptr;
2637:         if (lengthptr != NULL)
2638:         {
2639:             if (OFLOW_MAX - *lengthptr < code - last_code)
2640:             {
2641:                 *errorcodeptr = ERR20;
2642:                 goto FAILED;
2643:             }
2644:             *lengthptr += code - last_code; /* To include callout length */
2645:             DPRINTF((">> end branch \n"));
2646:         }
2647:         return TRUE;
2648:
2649:
=====*/
2650:
2651:     /* Handle single-character metacharacters. In multiline mode, ^ disables
2652:     the setting of any following char as a first character. */
2653:
2654:     case '^':
2655:         if ((options & PCRE_MULTILINE) != 0)

```

```

2656:  {
2657:    if (firstbyte == REQ_UNSET) firstbyte = REQ_NONE;
2658:  }
2659:  previous = NULL;
2660:  *code++ = OP_CIRC;
2661:  break;
2662:
2663:  case '$':
2664:    previous = NULL;
2665:    *code++ = OP_DOLL;
2666:    break;
2667:
2668:  /* There can never be a first char if '.' is first, whatever happens about
2669:  repeats. The value of reqbyte doesn't change either. */
2670:
2671:  case '.':
2672:    if (firstbyte == REQ_UNSET) firstbyte = REQ_NONE;
2673:    zerofirstbyte = firstbyte;
2674:    zeroreqbyte = reqbyte;
2675:    previous = code;
2676:    *code++ = ((options & PCRE_DOTALL) != 0)? OP_ALLANY: OP_ANY;
2677:    break;
2678:
2679:
2680:
=====
2681:  /* Character classes. If the included characters are all < 256, we build a
2682:  32-byte bitmap of the permitted characters, except in the special case
2683:  where there is only one such character. For negated classes, we build the
2684:  map as usual, then invert it at the end. However, we use a different opcode
2685:  so that data characters > 255 can be handled correctly.
2686:
2687:  If the class contains characters outside the 0-255 range, a different
2688:  opcode is compiled. It may optionally have a bit map for characters < 256,
2689:  but those above are explicitly listed afterwards. A flag byte tells
2690:  whether the bitmap is present, and whether this is a negated class or not.
2691:
2692:  In JavaScript compatibility mode, an isolated ']' causes an error. In
2693:  default (Perl) mode, it is treated as a data character. */
2694:
2695:  case ']':

```

```

2696: if ((cd->external_options & PCRE_JAVASCRIPT_COMPAT) != 0)
2697: {
2698:     *errorcodeptr = ERR64;
2699:     goto FAILED;
2700: }
2701: goto NORMAL_CHAR;
2702:
2703: case '[':
2704:     previous = code;
2705:
2706:     /* PCRE supports POSIX class stuff inside a class. Perl gives an error if
2707:     they are encountered at the top level, so we'll do that too. */
2708:
2709:     if ((ptr[1] == ':' || ptr[1] == '.' || ptr[1] == '=') &&
2710:         check_posix_syntax(ptr, &temp_ptr))
2711:     {
2712:         *errorcodeptr = (ptr[1] == ':')? ERR13 : ERR31;
2713:         goto FAILED;
2714:     }
2715:
2716:     /* If the first character is '^', set the negation flag and skip it. Also,
2717:     if the first few characters (either before or after ^) are \Q \E or \E we
2718:     skip them too. This makes for compatibility with Perl. */
2719:
2720:     negate_class = FALSE;
2721:     for (;;)
2722:     {
2723:         c = *(++ptr);
2724:         if (c == '\\')
2725:         {
2726:             if (ptr[1] == 'E') ptr++;
2727:             else if (strncmp((const char *)ptr+1, "Q\\E", 3) == 0) ptr += 3;
2728:             else break;
2729:         }
2730:         else if (!negate_class && c == '^')
2731:             negate_class = TRUE;
2732:         else break;
2733:     }
2734:
2735:     /* Empty classes are allowed in JavaScript compatibility mode. Otherwise,
2736:     an initial ']' is taken as a data character -- the code below handles

```

```

2737:  that. In JS mode, [] must always fail, so generate OP_FAIL, whereas
2738:  [^] must match any character, so generate OP_ALLANY. */
2739:
2740:  if (c ==']' && (cd->external_options & PCRE_JAVASCRIPT_COMPAT) != 0)
2741:  {
2742:    *code++ = negate_class? OP_ALLANY : OP_FAIL;
2743:    if (firstbyte == REQ_UNSET) firstbyte = REQ_NONE;
2744:    zerofirstbyte = firstbyte;
2745:    break;
2746:  }
2747:
2748:  /* If a class contains a negative special such as \S, we need to flip the
2749:  negation flag at the end, so that support for characters > 255 works
2750:  correctly (they are all included in the class). */
2751:
2752:  should_flip_negation = FALSE;
2753:
2754:  /* Keep a count of chars with values < 256 so that we can optimize the case
2755:  of just a single character (as long as it's < 256). However, For higher
2756:  valued UTF-8 characters, we don't yet do any optimization. */
2757:
2758:  class_charcount = 0;
2759:  class_lastchar = -1;
2760:
2761:  /* Initialize the 32-char bit map to all zeros. We build the map in a
2762:  temporary bit of memory, in case the class contains only 1 character (less
2763:  than 256), because in that case the compiled code doesn't use the bit map.
2764:  */
2765:
2766:  memset(classbits, 0, 32 * sizeof(uchar));
2767:
2768: #ifdef SUPPORT_UTF8
2769:   class_utf8 = FALSE;          /* No chars >= 256 */
2770:   class_utf8data = code + LINK_SIZE + 2; /* For UTF-8 items */
2771:   class_utf8data_base = class_utf8data; /* For resetting in pass 1 */
2772: #endif
2773:
2774:  /* Process characters until ] is reached. By writing this as a "do" it
2775:  means that an initial ] is taken as a data character. At the start of the
2776:  loop, c contains the first byte of the character. */
2777:

```

```

2778:  if (c != 0) do
2779:      {
2780:          const uschar *oldptr;
2781:
2782: #ifdef SUPPORT_UTF8
2783:     if (utf8 && c > 127)
2784:         { /* Braces are required because the */
2785:             GETCHARLEN(c, ptr, ptr); /* macro generates multiple statements */
2786:         }
2787:
2788:     /* In the pre-compile phase, accumulate the length of any UTF-8 extra
2789:     data and reset the pointer. This is so that very large classes that
2790:     contain a zillion UTF-8 characters no longer overwrite the work space
2791:     (which is on the stack). */
2792:
2793:     if (lengthptr != NULL)
2794:         {
2795:             *lengthptr += class_utf8data - class_utf8data_base;
2796:             class_utf8data = class_utf8data_base;
2797:         }
2798:
2799: #endif
2800:
2801:     /* Inside \Q... \E everything is literal except \E */
2802:
2803:     if (inescq)
2804:         {
2805:             if (c == '\\' && ptr[1] == 'E') /* If we are at \E */
2806:                 {
2807:                     inescq = FALSE; /* Reset literal state */
2808:                     ptr++; /* Skip the 'E' */
2809:                     continue; /* Carry on with next */
2810:                 }
2811:             goto CHECK_RANGE; /* Could be range if \E follows */
2812:         }
2813:
2814:     /* Handle POSIX class names. Perl allows a negation extension of the
2815:     form [^name:]. A square bracket that doesn't match the syntax is
2816:     treated as a literal. We also recognize the POSIX constructions
2817:     [.ch.] and [=ch=] ("collating elements") and fault them, as Perl
2818:     5.6 and 5.8 do. */

```

```

2819:
2820: if (c == '[' &&
2821:     (ptr[1] == ':' || ptr[1] == '.' || ptr[1] == '=')) &&
2822:     check_posix_syntax(ptr, &temp_ptr))
2823: {
2824:     BOOL local_negate = FALSE;
2825:     int posix_class, taboffset, tabopt;
2826:     register const uschar *cbits = cd->cbits;
2827:     uschar pbits[32];
2828:
2829:     if (ptr[1] != ':')
2830:     {
2831:         *errorcode_ptr = ERR31;
2832:         goto FAILED;
2833:     }
2834:
2835:     ptr += 2;
2836:     if (*ptr == '^')
2837:     {
2838:         local_negate = TRUE;
2839:         should_flip_negation = TRUE; /* Note negative special */
2840:         ptr++;
2841:     }
2842:
2843:     posix_class = check_posix_name(ptr, temp_ptr - ptr);
2844:     if (posix_class < 0)
2845:     {
2846:         *errorcode_ptr = ERR30;
2847:         goto FAILED;
2848:     }
2849:
2850:     /* If matching is caseless, upper and lower are converted to
2851:     alpha. This relies on the fact that the class table starts with
2852:     alpha, lower, upper as the first 3 entries. */
2853:
2854:     if ((options & PCRE_CASELESS) != 0 && posix_class <= 2)
2855:         posix_class = 0;
2856:
2857:     /* We build the bit map for the POSIX class in a chunk of local store
2858:     because we may be adding and subtracting from it, and we don't want to
2859:     subtract bits that may be in the main map already. At the end we or the

```



```

2860:     result into the bit map that is being built. */
2861:
2862:     posix_class *= 3;
2863:
2864:     /* Copy in the first table (always present) */
2865:
2866:     memcpy(pbits, cbits + posix_class_maps[posix_class],
2867:         32 * sizeof(uchar));
2868:
2869:     /* If there is a second table, add or remove it as required. */
2870:
2871:     taboffset = posix_class_maps[posix_class + 1];
2872:     tabopt = posix_class_maps[posix_class + 2];
2873:
2874:     if (taboffset >= 0)
2875:     {
2876:         if (tabopt >= 0)
2877:             for (c = 0; c < 32; c++) pbits[c] |= cbits[c + taboffset];
2878:         else
2879:             for (c = 0; c < 32; c++) pbits[c] &= ~cbits[c + taboffset];
2880:     }
2881:
2882:     /* Not see if we need to remove any special characters. An option
2883:     value of 1 removes vertical space and 2 removes underscore. */
2884:
2885:     if (tabopt < 0) tabopt = -tabopt;
2886:     if (tabopt == 1) pbits[1] &= ~0x3c;
2887:     else if (tabopt == 2) pbits[11] &= 0x7f;
2888:
2889:     /* Add the POSIX table or its complement into the main table that is
2890:     being built and we are done. */
2891:
2892:     if (local_negate)
2893:         for (c = 0; c < 32; c++) classbits[c] |= ~pbits[c];
2894:     else
2895:         for (c = 0; c < 32; c++) classbits[c] |= pbits[c];
2896:
2897:     ptr = temp_ptr + 1;
2898:     class_charcount = 10; /* Set > 1; assumes more than 1 per class */
2899:     continue; /* End of POSIX syntax handling */
2900: }

```

```

2901:
2902:  /* Backslash may introduce a single character, or it may introduce one
2903:  of the specials, which just set a flag. The sequence \b is a special
2904:  case. Inside a class (and only there) it is treated as backspace.
2905:  Elsewhere it marks a word boundary. Other escapes have preset maps ready
2906:  to 'or' into the one we are building. We assume they have more than one
2907:  character in them, so set class_charcount bigger than one. */
2908:
2909:  if (c == '\\')
2910:  {
2911:      c = check_escape(&ptr, errorcodeptr, cd->bracount, options, TRUE);
2912:      if (*errorcodeptr != 0) goto FAILED;
2913:
2914:      if (-c == ESC_b) c = '\b';    /* \b is backspace in a class */
2915:      else if (-c == ESC_X) c = 'X'; /* \X is literal X in a class */
2916:      else if (-c == ESC_R) c = 'R'; /* \R is literal R in a class */
2917:      else if (-c == ESC_Q)        /* Handle start of quoted string */
2918:      {
2919:          if (ptr[1] == '\\ && ptr[2] == 'E')
2920:          {
2921:              ptr += 2; /* avoid empty string */
2922:          }
2923:          else inescq = TRUE;
2924:          continue;
2925:      }
2926:      else if (-c == ESC_E) continue; /* Ignore orphan \E */
2927:
2928:      if (c < 0)
2929:      {
2930:          register const uschar *cbits = cd->cbits;
2931:          class_charcount += 2; /* Greater than 1 is what matters */
2932:
2933:          /* Save time by not doing this in the pre-compile phase. */
2934:
2935:          if (lengthptr == NULL) switch (-c)
2936:          {
2937:              case ESC_d:
2938:                  for (c = 0; c < 32; c++) classbits[c] |= cbits[c+cbit_digit];
2939:                  continue;
2940:
2941:              case ESC_D:

```

```

2942:     should_flip_negation = TRUE;
2943:     for (c = 0; c < 32; c++) classbits[c] |= ~cbits[c+cbit_digit];
2944:     continue;
2945:
2946:     case ESC_w:
2947:     for (c = 0; c < 32; c++) classbits[c] |= cbits[c+cbit_word];
2948:     continue;
2949:
2950:     case ESC_W:
2951:     should_flip_negation = TRUE;
2952:     for (c = 0; c < 32; c++) classbits[c] |= ~cbits[c+cbit_word];
2953:     continue;
2954:
2955:     case ESC_s:
2956:     for (c = 0; c < 32; c++) classbits[c] |= cbits[c+cbit_space];
2957:     classbits[1] &= ~0x08; /* Perl 5.004 onwards omits VT from \s */
2958:     continue;
2959:
2960:     case ESC_S:
2961:     should_flip_negation = TRUE;
2962:     for (c = 0; c < 32; c++) classbits[c] |= ~cbits[c+cbit_space];
2963:     classbits[1] |= 0x08; /* Perl 5.004 onwards omits VT from \s */
2964:     continue;
2965:
2966:     default: /* Not recognized; fall through */
2967:     break; /* Need "default" setting to stop compiler warning. */
2968:     }
2969:
2970: /* In the pre-compile phase, just do the recognition. */
2971:
2972: else if (c == -ESC_d || c == -ESC_D || c == -ESC_w ||
2973:         c == -ESC_W || c == -ESC_s || c == -ESC_S) continue;
2974:
2975: /* We need to deal with \H, \h, \V, and \v in both phases because
2976: they use extra memory. */
2977:
2978: if (-c == ESC_h)
2979: {
2980:     SETBIT(classbits, 0x09); /* VT */
2981:     SETBIT(classbits, 0x20); /* SPACE */
2982:     SETBIT(classbits, 0xa0); /* NSBP */

```

```

2983: #ifdef SUPPORT_UTF8
2984:     if (utf8)
2985:     {
2986:         class_utf8 = TRUE;
2987:         *class_utf8data++ = XCL_SINGLE;
2988:         class_utf8data += _pcre_ord2utf8(0x1680, class_utf8data);
2989:         *class_utf8data++ = XCL_SINGLE;
2990:         class_utf8data += _pcre_ord2utf8(0x180e, class_utf8data);
2991:         *class_utf8data++ = XCL_RANGE;
2992:         class_utf8data += _pcre_ord2utf8(0x2000, class_utf8data);
2993:         class_utf8data += _pcre_ord2utf8(0x200A, class_utf8data);
2994:         *class_utf8data++ = XCL_SINGLE;
2995:         class_utf8data += _pcre_ord2utf8(0x202f, class_utf8data);
2996:         *class_utf8data++ = XCL_SINGLE;
2997:         class_utf8data += _pcre_ord2utf8(0x205f, class_utf8data);
2998:         *class_utf8data++ = XCL_SINGLE;
2999:         class_utf8data += _pcre_ord2utf8(0x3000, class_utf8data);
3000:     }
3001: #endif
3002:     continue;
3003: }
3004:
3005: if (-c == ESC_H)
3006: {
3007:     for (c = 0; c < 32; c++)
3008:     {
3009:         int x = 0xff;
3010:         switch (c)
3011:         {
3012:             case 0x09/8: x ^= 1 << (0x09%8); break;
3013:             case 0x20/8: x ^= 1 << (0x20%8); break;
3014:             case 0xa0/8: x ^= 1 << (0xa0%8); break;
3015:             default: break;
3016:         }
3017:         classbits[c] |= x;
3018:     }
3019:
3020: #ifdef SUPPORT_UTF8
3021:     if (utf8)
3022:     {
3023:         class_utf8 = TRUE;

```

```

3024:     *class_utf8data++ = XCL_RANGE;
3025:     class_utf8data += _pcre_ord2utf8(0x0100, class_utf8data);
3026:     class_utf8data += _pcre_ord2utf8(0x167f, class_utf8data);
3027:     *class_utf8data++ = XCL_RANGE;
3028:     class_utf8data += _pcre_ord2utf8(0x1681, class_utf8data);
3029:     class_utf8data += _pcre_ord2utf8(0x180d, class_utf8data);
3030:     *class_utf8data++ = XCL_RANGE;
3031:     class_utf8data += _pcre_ord2utf8(0x180f, class_utf8data);
3032:     class_utf8data += _pcre_ord2utf8(0x1fff, class_utf8data);
3033:     *class_utf8data++ = XCL_RANGE;
3034:     class_utf8data += _pcre_ord2utf8(0x200B, class_utf8data);
3035:     class_utf8data += _pcre_ord2utf8(0x202e, class_utf8data);
3036:     *class_utf8data++ = XCL_RANGE;
3037:     class_utf8data += _pcre_ord2utf8(0x2030, class_utf8data);
3038:     class_utf8data += _pcre_ord2utf8(0x205e, class_utf8data);
3039:     *class_utf8data++ = XCL_RANGE;
3040:     class_utf8data += _pcre_ord2utf8(0x2060, class_utf8data);
3041:     class_utf8data += _pcre_ord2utf8(0x2fff, class_utf8data);
3042:     *class_utf8data++ = XCL_RANGE;
3043:     class_utf8data += _pcre_ord2utf8(0x3001, class_utf8data);
3044:     class_utf8data += _pcre_ord2utf8(0x7fffffff, class_utf8data);
3045: }
3046: #endif
3047:     continue;
3048: }
3049:
3050: if (-c == ESC_v)
3051: {
3052:     SETBIT(classbits, 0x0a); /* LF */
3053:     SETBIT(classbits, 0x0b); /* VT */
3054:     SETBIT(classbits, 0x0c); /* FF */
3055:     SETBIT(classbits, 0x0d); /* CR */
3056:     SETBIT(classbits, 0x85); /* NEL */
3057: #ifdef SUPPORT_UTF8
3058:     if (utf8)
3059:     {
3060:         class_utf8 = TRUE;
3061:         *class_utf8data++ = XCL_RANGE;
3062:         class_utf8data += _pcre_ord2utf8(0x2028, class_utf8data);
3063:         class_utf8data += _pcre_ord2utf8(0x2029, class_utf8data);
3064:     }

```

```

3065: #endif
3066:     continue;
3067: }
3068:
3069: if (-c == ESC_V)
3070: {
3071:     for (c = 0; c < 32; c++)
3072:     {
3073:         int x = 0xff;
3074:         switch (c)
3075:         {
3076:             case 0x0a/8: x ^= 1 << (0x0a%8);
3077:                 x ^= 1 << (0x0b%8);
3078:                 x ^= 1 << (0x0c%8);
3079:                 x ^= 1 << (0x0d%8);
3080:                 break;
3081:             case 0x85/8: x ^= 1 << (0x85%8); break;
3082:             default: break;
3083:         }
3084:         classbits[c] |= x;
3085:     }
3086:
3087: #ifdef SUPPORT_UTF8
3088:     if (utf8)
3089:     {
3090:         class_utf8 = TRUE;
3091:         *class_utf8data++ = XCL_RANGE;
3092:         class_utf8data += _pcre_ord2utf8(0x0100, class_utf8data);
3093:         class_utf8data += _pcre_ord2utf8(0x2027, class_utf8data);
3094:         *class_utf8data++ = XCL_RANGE;
3095:         class_utf8data += _pcre_ord2utf8(0x2029, class_utf8data);
3096:         class_utf8data += _pcre_ord2utf8(0x7fffffff, class_utf8data);
3097:     }
3098: #endif
3099:     continue;
3100: }
3101:
3102: /* We need to deal with \P and \p in both phases. */
3103:
3104: #ifdef SUPPORT_UCP
3105:     if (-c == ESC_p || -c == ESC_P)

```

```

3106:     {
3107:     BOOL negated;
3108:     int pdata;
3109:     int ptype = get_ucp(&ptr, &negated, &pdata, errorcodeptr);
3110:     if (ptype < 0) goto FAILED;
3111:     class_utf8 = TRUE;
3112:     *class_utf8data++ = ((-c == ESC_p) != negated)?
3113:         XCL_PROP : XCL_NOTPROP;
3114:     *class_utf8data++ = ptype;
3115:     *class_utf8data++ = pdata;
3116:     class_charcount -= 2; /* Not a < 256 character */
3117:     continue;
3118:     }
3119: #endif
3120:     /* Unrecognized escapes are faulted if PCRE is running in its
3121:     strict mode. By default, for compatibility with Perl, they are
3122:     treated as literals. */
3123:
3124:     if ((options & PCRE_EXTRA) != 0)
3125:     {
3126:         *errorcodeptr = ERR7;
3127:         goto FAILED;
3128:     }
3129:
3130:     class_charcount -= 2; /* Undo the default count from above */
3131:     c = *ptr;          /* Get the final character and fall through */
3132:     }
3133:
3134:     /* Fall through if we have a single character (c >= 0). This may be
3135:     greater than 256 in UTF-8 mode. */
3136:
3137:     } /* End of backslash handling */
3138:
3139:     /* A single character may be followed by '-' to form a range. However,
3140:     Perl does not permit ']' to be the end of the range. A '-' character
3141:     at the end is treated as a literal. Perl ignores orphaned \E sequences
3142:     entirely. The code for handling \Q and \E is messy. */
3143:
3144:     CHECK_RANGE:
3145:     while (ptr[1] == '\\' && ptr[2] == 'E')
3146:     {

```

```

3147:     inescq = FALSE;
3148:     ptr += 2;
3149: }
3150:
3151: oldptr = ptr;
3152:
3153: /* Remember \r or \n */
3154:
3155: if (c == '\r' || c == '\n') cd->external_flags |= PCRE_HASCORRLF;
3156:
3157: /* Check for range */
3158:
3159: if (!inescq && ptr[1] == '-')
3160: {
3161:     int d;
3162:     ptr += 2;
3163:     while (*ptr == '\\' && ptr[1] == 'E') ptr += 2;
3164:
3165:     /* If we hit \Q (not followed by \E) at this point, go into escaped
3166:     mode. */
3167:
3168:     while (*ptr == '\\' && ptr[1] == 'Q')
3169:     {
3170:         ptr += 2;
3171:         if (*ptr == '\\' && ptr[1] == 'E') { ptr += 2; continue; }
3172:         inescq = TRUE;
3173:         break;
3174:     }
3175:
3176:     if (*ptr == 0 || (!inescq && *ptr == ']'))
3177:     {
3178:         ptr = oldptr;
3179:         goto LONE_SINGLE_CHARACTER;
3180:     }
3181:
3182: #ifdef SUPPORT_UTF8
3183:     if (utf8)
3184:     {
3185:         /* Braces are required because the */
3186:         GETCHARLEN(d, ptr, ptr); /* macro generates multiple statements */
3187:     }
3188:     else

```



```

3188: #endif
3189:     d = *ptr; /* Not UTF-8 mode */
3190:
3191:     /* The second part of a range can be a single-character escape, but
3192:     not any of the other escapes. Perl 5.6 treats a hyphen as a literal
3193:     in such circumstances. */
3194:
3195:     if (!linescq && d == '\\')
3196:     {
3197:         d = check_escape(&ptr, errorcodeptr, cd->bracount, options, TRUE);
3198:         if (*errorcodeptr != 0) goto FAILED;
3199:
3200:         /* \b is backspace; \X is literal X; \R is literal R; any other
3201:         special means the '-' was literal */
3202:
3203:         if (d < 0)
3204:         {
3205:             if (d == -ESC_b) d = '\b';
3206:             else if (d == -ESC_X) d = 'X';
3207:             else if (d == -ESC_R) d = 'R'; else
3208:             {
3209:                 ptr = oldptr;
3210:                 goto LONE_SINGLE_CHARACTER; /* A few lines below */
3211:             }
3212:         }
3213:     }
3214:
3215:     /* Check that the two values are in the correct order. Optimize
3216:     one-character ranges */
3217:
3218:     if (d < c)
3219:     {
3220:         *errorcodeptr = ERR8;
3221:         goto FAILED;
3222:     }
3223:
3224:     if (d == c) goto LONE_SINGLE_CHARACTER; /* A few lines below */
3225:
3226:     /* Remember \r or \n */
3227:
3228:     if (d == '\r' || d == '\n') cd->external_flags |= PCRE_HASCORRLF;

```

```

3229:
3230:     /* In UTF-8 mode, if the upper limit is > 255, or > 127 for caseless
3231:     matching, we have to use an XCLASS with extra data items. Caseless
3232:     matching for characters > 127 is available only if UCP support is
3233:     available. */
3234:
3235: #ifdef SUPPORT_UTF8
3236:     if (utf8 && (d > 255 || ((options & PCRE_CASELESS) != 0 && d > 127)))
3237:     {
3238:         class_utf8 = TRUE;
3239:
3240:         /* With UCP support, we can find the other case equivalents of
3241:         the relevant characters. There may be several ranges. Optimize how
3242:         they fit with the basic range. */
3243:
3244: #ifdef SUPPORT_UCP
3245:         if ((options & PCRE_CASELESS) != 0)
3246:         {
3247:             unsigned int occ, ocd;
3248:             unsigned int cc = c;
3249:             unsigned int origd = d;
3250:             while (get_othercase_range(&cc, origd, &occ, &ocd))
3251:             {
3252:                 if (occ >= (unsigned int)c &&
3253:                     ocd <= (unsigned int)d)
3254:                     continue;          /* Skip embedded ranges */
3255:
3256:                 if (occ < (unsigned int)c &&
3257:                     ocd >= (unsigned int)c - 1) /* Extend the basic range */
3258:                 {
3259:                     /* if there is overlap, */
3259:                     c = occ;          /* noting that if occ < c */
3260:                     continue;        /* we can't have ocd > d */
3261:                 }
3261:                 /* because a subrange is */
3262:                 if (ocd > (unsigned int)d &&
3263:                     occ <= (unsigned int)d + 1) /* always shorter than */
3264:                 {
3264:                     /* the basic range. */
3265:                     d = ocd;
3266:                     continue;
3267:                 }
3268:
3269:                 if (occ == ocd)

```

```

3270:     {
3271:         *class_utf8data++ = XCL_SINGLE;
3272:     }
3273:     else
3274:     {
3275:         *class_utf8data++ = XCL_RANGE;
3276:         class_utf8data += _pcre_ord2utf8(occ, class_utf8data);
3277:     }
3278:     class_utf8data += _pcre_ord2utf8(ocd, class_utf8data);
3279: }
3280: }
3281: #endif /* SUPPORT_UCP */
3282:
3283: /* Now record the original range, possibly modified for UCP caseless
3284: overlapping ranges. */
3285:
3286: *class_utf8data++ = XCL_RANGE;
3287: class_utf8data += _pcre_ord2utf8(c, class_utf8data);
3288: class_utf8data += _pcre_ord2utf8(d, class_utf8data);
3289:
3290: /* With UCP support, we are done. Without UCP support, there is no
3291: caseless matching for UTF-8 characters > 127; we can use the bit map
3292: for the smaller ones. */
3293:
3294: #ifdef SUPPORT_UCP
3295:     continue; /* With next character in the class */
3296: #else
3297:     if ((options & PCRE_CASELESS) == 0 || c > 127) continue;
3298:
3299:     /* Adjust upper limit and fall through to set up the map */
3300:
3301:     d = 127;
3302:
3303: #endif /* SUPPORT_UCP */
3304: }
3305: #endif /* SUPPORT_UTF8 */
3306:
3307: /* We use the bit map for all cases when not in UTF-8 mode; else
3308: ranges that lie entirely within 0-127 when there is UCP support; else
3309: for partial ranges without UCP support. */
3310:

```

```

3311:    class_charcount += d - c + 1;
3312:    class_lastchar = d;
3313:
3314:    /* We can save a bit of time by skipping this in the pre-compile. */
3315:
3316:    if (lengthptr == NULL) for (; c <= d; c++)
3317:    {
3318:        classbits[c/8] |= (1 << (c&7));
3319:        if ((options & PCRE_CASELESS) != 0)
3320:        {
3321:            int uc = cd->fcc[c];      /* flip case */
3322:            classbits[uc/8] |= (1 << (uc&7));
3323:        }
3324:    }
3325:
3326:    continue; /* Go get the next char in the class */
3327: }
3328:
3329: /* Handle a lone single character - we can get here for a normal
3330: non-escape char, or after \ that introduces a single character or for an
3331: apparent range that isn't. */
3332:
3333: LONE_SINGLE_CHARACTER:
3334:
3335: /* Handle a character that cannot go in the bit map */
3336:
3337: #ifdef SUPPORT_UTF8
3338:     if (utf8 && (c > 255 || ((options & PCRE_CASELESS) != 0 && c > 127)))
3339:     {
3340:         class_utf8 = TRUE;
3341:         *class_utf8data++ = XCL_SINGLE;
3342:         class_utf8data += _pcre_ord2utf8(c, class_utf8data);
3343:
3344: #ifdef SUPPORT_UCP
3345:         if ((options & PCRE_CASELESS) != 0)
3346:         {
3347:             unsigned int othercase;
3348:             if ((othercase = UCD_OTHERCASE(c)) != c)
3349:             {
3350:                 *class_utf8data++ = XCL_SINGLE;
3351:                 class_utf8data += _pcre_ord2utf8(othercase, class_utf8data);

```

```

3352:     }
3353: }
3354: #endif /* SUPPORT_UCP */
3355:
3356:     }
3357:     else
3358: #endif /* SUPPORT_UTF8 */
3359:
3360:     /* Handle a single-byte character */
3361:     {
3362:         classbits[c/8] |= (1 << (c&7));
3363:         if ((options & PCRE_CASELESS) != 0)
3364:         {
3365:             c = cd->fcc[c]; /* flip case */
3366:             classbits[c/8] |= (1 << (c&7));
3367:         }
3368:         class_charcount++;
3369:         class_lastchar = c;
3370:     }
3371: }
3372:
3373: /* Loop until ']' reached. This "while" is the end of the "do" above. */
3374:
3375: while ((c = *(++ptr)) != 0 && (c != ']' || inescq));
3376:
3377: if (c == 0) /* Missing terminating ']' */
3378: {
3379:     *errorcodeptr = ERR6;
3380:     goto FAILED;
3381: }
3382:
3383:
3384: /* This code has been disabled because it would mean that \s counts as
3385: an explicit \r or \n reference, and that's not really what is wanted. Now
3386: we set the flag only if there is a literal "\r" or "\n" in the class. */
3387:
3388: #if 0
3389:     /* Remember whether \r or \n are in this class */
3390:
3391:     if (negate_class)
3392:     {

```

```

3393:   if ((classbits[1] & 0x24) != 0x24) cd->external_flags |= PCRE_HASCORLRF;
3394:   }
3395:   else
3396:   {
3397:     if ((classbits[1] & 0x24) != 0) cd->external_flags |= PCRE_HASCORLRF;
3398:   }
3399: #endif
3400:
3401:
3402:   /* If class_charcount is 1, we saw precisely one character whose value is
3403:   less than 256. As long as there were no characters >= 128 and there was no
3404:   use of \p or \P, in other words, no use of any XCLASS features, we can
3405:   optimize.
3406:
3407:   In UTF-8 mode, we can optimize the negative case only if there were no
3408:   characters >= 128 because OP_NOT and the related opcodes like OP_NOTSTAR
3409:   operate on single-bytes only. This is an historical hangover. Maybe one day
3410:   we can tidy these opcodes to handle multi-byte characters.
3411:
3412:   The optimization throws away the bit map. We turn the item into a
3413:   1-character OP_CHAR[NC] if it's positive, or OP_NOT if it's negative. Note
3414:   that OP_NOT does not support multibyte characters. In the positive case, it
3415:   can cause firstbyte to be set. Otherwise, there can be no first char if
3416:   this item is first, whatever repeat count may follow. In the case of
3417:   reqbyte, save the previous value for reinstating. */
3418:
3419: #ifdef SUPPORT_UTF8
3420:   if (class_charcount == 1 && !class_utf8 &&
3421:       (!utf8 || !negate_class || class_lastchar < 128))
3422:   #else
3423:   if (class_charcount == 1)
3424:   #endif
3425:   {
3426:     zeroreqbyte = reqbyte;
3427:
3428:     /* The OP_NOT opcode works on one-byte characters only. */
3429:
3430:     if (negate_class)
3431:     {
3432:       if (firstbyte == REQ_UNSET) firstbyte = REQ_NONE;
3433:       zerofirstbyte = firstbyte;

```

```

3434:     *code++ = OP_NOT;
3435:     *code++ = class_lastchar;
3436:     break;
3437: }
3438:
3439: /* For a single, positive character, get the value into mcbuffer, and
3440: then we can handle this with the normal one-character code. */
3441:
3442: #ifdef SUPPORT_UTF8
3443:     if (utf8 && class_lastchar > 127)
3444:         mclength = _pcre_ord2utf8(class_lastchar, mcbuffer);
3445:     else
3446: #endif
3447:     {
3448:         mcbuffer[0] = class_lastchar;
3449:         mclength = 1;
3450:     }
3451:     goto ONE_CHAR;
3452: } /* End of 1-char optimization */
3453:
3454: /* The general case - not the one-char optimization. If this is the first
3455: thing in the branch, there can be no first char setting, whatever the
3456: repeat count. Any reqbyte setting must remain unchanged after any kind of
3457: repeat. */
3458:
3459: if (firstbyte == REQ_UNSET) firstbyte = REQ_NONE;
3460: zerofirstbyte = firstbyte;
3461: zeroreqbyte = reqbyte;
3462:
3463: /* If there are characters with values > 255, we have to compile an
3464: extended class, with its own opcode, unless there was a negated special
3465: such as \S in the class, because in that case all characters > 255 are in
3466: the class, so any that were explicitly given as well can be ignored. If
3467: (when there are explicit characters > 255 that must be listed) there are no
3468: characters < 256, we can omit the bitmap in the actual compiled code. */
3469:
3470: #ifdef SUPPORT_UTF8
3471:     if (class_utf8 && !should_flip_negation)
3472:     {
3473:         *class_utf8data++ = XCL_END; /* Marks the end of extra data */
3474:         *code++ = OP_XCLASS;

```

```

3475: code += LINK_SIZE;
3476: *code = negate_class? XCL_NOT : 0;
3477:
3478: /* If the map is required, move up the extra data to make room for it;
3479: otherwise just move the code pointer to the end of the extra data. */
3480:
3481: if (class_charcount > 0)
3482: {
3483: *code++ |= XCL_MAP;
3484: memmove(code + 32, code, class_utf8data - code);
3485: memcpy(code, classbits, 32);
3486: code = class_utf8data + 32;
3487: }
3488: else code = class_utf8data;
3489:
3490: /* Now fill in the complete length of the item */
3491:
3492: PUT(previous, 1, code - previous);
3493: break; /* End of class handling */
3494: }
3495: #endif
3496:
3497: /* If there are no characters > 255, set the opcode to OP_CLASS or
3498: OP_NCLASS, depending on whether the whole class was negated and whether
3499: there were negative specials such as \S in the class. Then copy the 32-byte
3500: map into the code vector, negating it if necessary. */
3501:
3502: *code++ = (negate_class == should_flip_negation) ? OP_CLASS : OP_NCLASS;
3503: if (negate_class)
3504: {
3505: if (lengthptr == NULL) /* Save time in the pre-compile phase */
3506: for (c = 0; c < 32; c++) code[c] = ~classbits[c];
3507: }
3508: else
3509: {
3510: memcpy(code, classbits, 32);
3511: }
3512: code += 32;
3513: break;
3514:
3515:

```



```

3516:                                                                    /*
=====*/
3517:  /* Various kinds of repeat; '{' is not necessarily a quantifier, but this
3518:  has been tested above. */
3519:
3520:  case '{':
3521:  if (!is_quantifier) goto NORMAL_CHAR;
3522:  ptr = read_repeat_counts(ptr+1, &repeat_min, &repeat_max, errorcodeptr);
3523:  if (*errorcodeptr != 0) goto FAILED;
3524:  goto REPEAT;
3525:
3526:  case '*':
3527:  repeat_min = 0;
3528:  repeat_max = -1;
3529:  goto REPEAT;
3530:
3531:  case '+':
3532:  repeat_min = 1;
3533:  repeat_max = -1;
3534:  goto REPEAT;
3535:
3536:  case '?':
3537:  repeat_min = 0;
3538:  repeat_max = 1;
3539:
3540:  REPEAT:
3541:  if (previous == NULL)
3542:  {
3543:    *errorcodeptr = ERR9;
3544:    goto FAILED;
3545:  }
3546:
3547:  if (repeat_min == 0)
3548:  {
3549:    firstbyte = zerofirstbyte;  /* Adjust for zero repeat */
3550:    reqbyte = zeroreqbyte;     /* Ditto */
3551:  }
3552:
3553:  /* Remember whether this is a variable length repeat */
3554:
3555:  reqvary = (repeat_min == repeat_max)? 0 : REQ_VARY;

```

```

3556:
3557:  op_type = 0;          /* Default single-char op codes */
3558:  possessive_quantifier = FALSE; /* Default not possessive quantifier */
3559:
3560:  /* Save start of previous item, in case we have to move it up to make space
3561:  for an inserted OP_ONCE for the additional '+' extension. */
3562:
3563:  tempcode = previous;
3564:
3565:  /* If the next character is '+', we have a possessive quantifier. This
3566:  implies greediness, whatever the setting of the PCRE_UNGREEDY option.
3567:  If the next character is '?' this is a minimizing repeat, by default,
3568:  but if PCRE_UNGREEDY is set, it works the other way round. We change the
3569:  repeat type to the non-default. */
3570:
3571:  if (ptr[1] == '+')
3572:  {
3573:    repeat_type = 0;          /* Force greedy */
3574:    possessive_quantifier = TRUE;
3575:    ptr++;
3576:  }
3577:  else if (ptr[1] == '?')
3578:  {
3579:    repeat_type = greedy_non_default;
3580:    ptr++;
3581:  }
3582:  else repeat_type = greedy_default;
3583:
3584:  /* If previous was a character match, abolish the item and generate a
3585:  repeat item instead. If a char item has a minimum of more than one, ensure
3586:  that it is set in reqbyte - it might not be if a sequence such as x{3} is
3587:  the first thing in a branch because the x will have gone into firstbyte
3588:  instead. */
3589:
3590:  if (*previous == OP_CHAR || *previous == OP_CHARNC)
3591:  {
3592:    /* Deal with UTF-8 characters that take up more than one byte. It's
3593:    easier to write this out separately than try to macrify it. Use c to
3594:    hold the length of the character in bytes, plus 0x80 to flag that it's a
3595:    length rather than a small character. */
3596:

```

```

3597: #ifdef SUPPORT_UTF8
3598:     if (utf8 && (code[-1] & 0x80) != 0)
3599:     {
3600:         uschar *lastchar = code - 1;
3601:         while((*lastchar & 0xc0) == 0x80) lastchar--;
3602:         c = code - lastchar;          /* Length of UTF-8 character */
3603:         memcpy(utf8_char, lastchar, c); /* Save the char */
3604:         c |= 0x80;                    /* Flag c as a length */
3605:     }
3606:     else
3607: #endif
3608:
3609:     /* Handle the case of a single byte - either with no UTF8 support, or
3610:     with UTF-8 disabled, or for a UTF-8 character < 128. */
3611:
3612:     {
3613:         c = code[-1];
3614:         if (repeat_min > 1) reqbyte = c | req_caseopt | cd->req_varyopt;
3615:     }
3616:
3617:     /* If the repetition is unlimited, it pays to see if the next thing on
3618:     the line is something that cannot possibly match this character. If so,
3619:     automatically possessifying this item gains some performance in the case
3620:     where the match fails. */
3621:
3622:     if (!possessive_quantifier &&
3623:         repeat_max < 0 &&
3624:         check_auto_possessive(*previous, c, utf8, utf8_char, ptr + 1,
3625:             options, cd))
3626:     {
3627:         repeat_type = 0; /* Force greedy */
3628:         possessive_quantifier = TRUE;
3629:     }
3630:
3631:     goto OUTPUT_SINGLE_REPEAT; /* Code shared with single character types */
3632: }
3633:
3634: /* If previous was a single negated character ([^a] or similar), we use
3635: one of the special opcodes, replacing it. The code is shared with single-
3636: character repeats by setting opt_type to add a suitable offset into
3637: repeat_type. We can also test for auto-possessification. OP_NOT is

```

```

3638: currently used only for single-byte chars. */
3639:
3640: else if (*previous == OP_NOT)
3641: {
3642:     op_type = OP_NOTSTAR - OP_STAR; /* Use "not" opcodes */
3643:     c = previous[1];
3644:     if (!possessive_quantifier &&
3645:         repeat_max < 0 &&
3646:         check_auto_possessive(OP_NOT, c, utf8, NULL, ptr + 1, options, cd))
3647:     {
3648:         repeat_type = 0; /* Force greedy */
3649:         possessive_quantifier = TRUE;
3650:     }
3651:     goto OUTPUT_SINGLE_REPEAT;
3652: }
3653:
3654: /* If previous was a character type match ( \d or similar), abolish it and
3655: create a suitable repeat item. The code is shared with single-character
3656: repeats by setting op_type to add a suitable offset into repeat_type. Note
3657: the the Unicode property types will be present only when SUPPORT_UCP is
3658: defined, but we don't wrap the little bits of code here because it just
3659: makes it horribly messy. */
3660:
3661: else if (*previous < OP_EODN)
3662: {
3663:     uschar *oldcode;
3664:     int prop_type, prop_value;
3665:     op_type = OP_TYPESTAR - OP_STAR; /* Use type opcodes */
3666:     c = *previous;
3667:
3668:     if (!possessive_quantifier &&
3669:         repeat_max < 0 &&
3670:         check_auto_possessive(c, 0, utf8, NULL, ptr + 1, options, cd))
3671:     {
3672:         repeat_type = 0; /* Force greedy */
3673:         possessive_quantifier = TRUE;
3674:     }
3675:
3676: OUTPUT_SINGLE_REPEAT:
3677: if (*previous == OP_PROP || *previous == OP_NOTPROP)
3678: {

```

```

3679:     prop_type = previous[1];
3680:     prop_value = previous[2];
3681: }
3682: else prop_type = prop_value = -1;
3683:
3684: oldcode = code;
3685: code = previous;          /* Usually overwrite previous item */
3686:
3687: /* If the maximum is zero then the minimum must also be zero; Perl allows
3688: this case, so we do too - by simply omitting the item altogether. */
3689:
3690: if (repeat_max == 0) goto END_REPEAT;
3691:
3692: /* All real repeats make it impossible to handle partial matching (maybe
3693: one day we will be able to remove this restriction). */
3694:
3695: if (repeat_max != 1) cd->external_flags |= PCRE_NOPARTIAL;
3696:
3697: /* Combine the op_type with the repeat_type */
3698:
3699: repeat_type += op_type;
3700:
3701: /* A minimum of zero is handled either as the special case * or ?, or as
3702: an UPTO, with the maximum given. */
3703:
3704: if (repeat_min == 0)
3705: {
3706:     if (repeat_max == -1) *code++ = OP_STAR + repeat_type;
3707:     else if (repeat_max == 1) *code++ = OP_QUERY + repeat_type;
3708:     else
3709:     {
3710:         *code++ = OP_UPTO + repeat_type;
3711:         PUT2INC(code, 0, repeat_max);
3712:     }
3713: }
3714:
3715: /* A repeat minimum of 1 is optimized into some special cases. If the
3716: maximum is unlimited, we use OP_PLUS. Otherwise, the original item is
3717: left in place and, if the maximum is greater than 1, we use OP_UPTO with
3718: one less than the maximum. */
3719:

```

```

3720:     else if (repeat_min == 1)
3721:     {
3722:         if (repeat_max == -1)
3723:             *code++ = OP_PLUS + repeat_type;
3724:         else
3725:         {
3726:             code = oldcode;          /* leave previous item in place */
3727:             if (repeat_max == 1) goto END_REPEAT;
3728:             *code++ = OP_UPTO + repeat_type;
3729:             PUT2INC(code, 0, repeat_max - 1);
3730:         }
3731:     }
3732:
3733:     /* The case {n,n} is just an EXACT, while the general case {n,m} is
3734:     handled as an EXACT followed by an UPTO. */
3735:
3736:     else
3737:     {
3738:         *code++ = OP_EXACT + op_type; /* NB EXACT doesn't have repeat_type */
3739:         PUT2INC(code, 0, repeat_min);
3740:
3741:         /* If the maximum is unlimited, insert an OP_STAR. Before doing so,
3742:         we have to insert the character for the previous code. For a repeated
3743:         Unicode property match, there are two extra bytes that define the
3744:         required property. In UTF-8 mode, long characters have their length in
3745:         c, with the 0x80 bit as a flag. */
3746:
3747:         if (repeat_max < 0)
3748:         {
3749: #ifdef SUPPORT_UTF8
3750:             if (utf8 && c >= 128)
3751:             {
3752:                 memcpy(code, utf8_char, c & 7);
3753:                 code += c & 7;
3754:             }
3755:         else
3756: #endif
3757:         {
3758:             *code++ = c;
3759:             if (prop_type >= 0)
3760:             {

```

```

3761:     *code++ = prop_type;
3762:     *code++ = prop_value;
3763: }
3764: }
3765: *code++ = OP_STAR + repeat_type;
3766: }
3767:
3768: /* Else insert an UPTO if the max is greater than the min, again
3769: preceded by the character, for the previously inserted code. If the
3770: UPTO is just for 1 instance, we can use QUERY instead. */
3771:
3772: else if (repeat_max != repeat_min)
3773: {
3774: #ifdef SUPPORT_UTF8
3775:     if (utf8 && c >= 128)
3776:     {
3777:         memcpy(code, utf8_char, c & 7);
3778:         code += c & 7;
3779:     }
3780:     else
3781: #endif
3782:     *code++ = c;
3783:     if (prop_type >= 0)
3784:     {
3785:         *code++ = prop_type;
3786:         *code++ = prop_value;
3787:     }
3788:     repeat_max -= repeat_min;
3789:
3790:     if (repeat_max == 1)
3791:     {
3792:         *code++ = OP_QUERY + repeat_type;
3793:     }
3794:     else
3795:     {
3796:         *code++ = OP_UPTO + repeat_type;
3797:         PUT2INC(code, 0, repeat_max);
3798:     }
3799: }
3800: }
3801:

```

```

3802:  /* The character or character type itself comes last in all cases. */
3803:
3804: #ifdef SUPPORT_UTF8
3805:     if (utf8 && c >= 128)
3806:     {
3807:         memcpy(code, utf8_char, c & 7);
3808:         code += c & 7;
3809:     }
3810:     else
3811: #endif
3812:     *code++ = c;
3813:
3814:  /* For a repeated Unicode property match, there are two extra bytes that
3815:     define the required property. */
3816:
3817: #ifdef SUPPORT_UCP
3818:     if (prop_type >= 0)
3819:     {
3820:         *code++ = prop_type;
3821:         *code++ = prop_value;
3822:     }
3823: #endif
3824: }
3825:
3826:  /* If previous was a character class or a back reference, we put the repeat
3827:     stuff after it, but just skip the item if the repeat was {0,0}. */
3828:
3829:     else if (*previous == OP_CLASS ||
3830:             *previous == OP_NCLASS ||
3831: #ifdef SUPPORT_UTF8
3832:             *previous == OP_XCLASS ||
3833: #endif
3834:             *previous == OP_REF)
3835:     {
3836:         if (repeat_max == 0)
3837:         {
3838:             code = previous;
3839:             goto END_REPEAT;
3840:         }
3841:
3842:  /* All real repeats make it impossible to handle partial matching (maybe

```



```

3843:     one day we will be able to remove this restriction). */
3844:
3845:     if (repeat_max != 1) cd->external_flags |= PCRE_NOPARTIAL;
3846:
3847:     if (repeat_min == 0 && repeat_max == -1)
3848:         *code++ = OP_CRSTAR + repeat_type;
3849:     else if (repeat_min == 1 && repeat_max == -1)
3850:         *code++ = OP_CRPLUS + repeat_type;
3851:     else if (repeat_min == 0 && repeat_max == 1)
3852:         *code++ = OP_CRQUERY + repeat_type;
3853:     else
3854:     {
3855:         *code++ = OP_CRRANGE + repeat_type;
3856:         PUT2INC(code, 0, repeat_min);
3857:         if (repeat_max == -1) repeat_max = 0; /* 2-byte encoding for max */
3858:         PUT2INC(code, 0, repeat_max);
3859:     }
3860: }
3861:
3862: /* If previous was a bracket group, we may have to replicate it in certain
3863: cases. */
3864:
3865: else if (*previous == OP_BRA || *previous == OP_CBRA ||
3866:         *previous == OP_ONCE || *previous == OP_COND)
3867: {
3868:     register int i;
3869:     int ketoffset = 0;
3870:     int len = code - previous;
3871:     uschar *bralink = NULL;
3872:
3873:     /* Repeating a DEFINE group is pointless */
3874:
3875:     if (*previous == OP_COND && previous[LINK_SIZE+1] == OP_DEF)
3876:     {
3877:         *errorcodeptr = ERR55;
3878:         goto FAILED;
3879:     }
3880:
3881:     /* If the maximum repeat count is unlimited, find the end of the bracket
3882: by scanning through from the start, and compute the offset back to it
3883: from the current code pointer. There may be an OP_OPT setting following

```

```

3884: the final KET, so we can't find the end just by going back from the code
3885: pointer. */
3886:
3887: if (repeat_max == -1)
3888: {
3889:     register uschar *ket = previous;
3890:     do ket += GET(ket, 1); while (*ket != OP_KET);
3891:     ketoffset = code - ket;
3892: }
3893:
3894: /* The case of a zero minimum is special because of the need to stick
3895: OP_BRAZERO in front of it, and because the group appears once in the
3896: data, whereas in other cases it appears the minimum number of times. For
3897: this reason, it is simplest to treat this case separately, as otherwise
3898: the code gets far too messy. There are several special subcases when the
3899: minimum is zero. */
3900:
3901: if (repeat_min == 0)
3902: {
3903:     /* If the maximum is also zero, we used to just omit the group from the
3904: output altogether, like this:
3905:
3906:     ** if (repeat_max == 0)
3907:     ** {
3908:     **     code = previous;
3909:     **     goto END_REPEAT;
3910:     ** }
3911:
3912: However, that fails when a group is referenced as a subroutine from
3913: elsewhere in the pattern, so now we stick in OP_SKIPZERO in front of it
3914: so that it is skipped on execution. As we don't have a list of which
3915: groups are referenced, we cannot do this selectively.
3916:
3917: If the maximum is 1 or unlimited, we just have to stick in the BRAZERO
3918: and do no more at this point. However, we do need to adjust any
3919: OP_RECURSE calls inside the group that refer to the group itself or any
3920: internal or forward referenced group, because the offset is from the
3921: start of the whole regex. Temporarily terminate the pattern while doing
3922: this. */
3923:
3924: if (repeat_max <= 1) /* Covers 0, 1, and unlimited */

```

```

3925:     {
3926:     *code = OP_END;
3927:     adjust_recurse(previous, 1, utf8, cd, save_hwm);
3928:     memmove(previous+1, previous, len);
3929:     code++;
3930:     if (repeat_max == 0)
3931:     {
3932:         *previous++ = OP_SKIPZERO;
3933:         goto END_REPEAT;
3934:     }
3935:     *previous++ = OP_BRAZERO + repeat_type;
3936: }
3937:
3938: /* If the maximum is greater than 1 and limited, we have to replicate
3939: in a nested fashion, sticking OP_BRAZERO before each set of brackets.
3940: The first one has to be handled carefully because it's the original
3941: copy, which has to be moved up. The remainder can be handled by code
3942: that is common with the non-zero minimum case below. We have to
3943: adjust the value of repeat_max, since one less copy is required. Once
3944: again, we may have to adjust any OP_RECURSE calls inside the group. */
3945:
3946: else
3947: {
3948:     int offset;
3949:     *code = OP_END;
3950:     adjust_recurse(previous, 2 + LINK_SIZE, utf8, cd, save_hwm);
3951:     memmove(previous + 2 + LINK_SIZE, previous, len);
3952:     code += 2 + LINK_SIZE;
3953:     *previous++ = OP_BRAZERO + repeat_type;
3954:     *previous++ = OP_BRA;
3955:
3956:     /* We chain together the bracket offset fields that have to be
3957: filled in later when the ends of the brackets are reached. */
3958:
3959:     offset = (bralink == NULL)? 0 : previous - bralink;
3960:     bralink = previous;
3961:     PUTINC(previous, 0, offset);
3962: }
3963:
3964: repeat_max--;
3965: }

```

```

3966:
3967:  /* If the minimum is greater than zero, replicate the group as many
3968:  times as necessary, and adjust the maximum to the number of subsequent
3969:  copies that we need. If we set a first char from the group, and didn't
3970:  set a required char, copy the latter from the former. If there are any
3971:  forward reference subroutine calls in the group, there will be entries on
3972:  the workspace list; replicate these with an appropriate increment. */
3973:
3974:  else
3975:  {
3976:    if (repeat_min > 1)
3977:    {
3978:      /* In the pre-compile phase, we don't actually do the replication. We
3979:      just adjust the length as if we had. Do some paranoid checks for
3980:      potential integer overflow. */
3981:
3982:      if (lengthptr != NULL)
3983:      {
3984:        int delta = (repeat_min - 1)*length_prevgroup;
3985:        if (((double)(repeat_min - 1)*(double)length_prevgroup >
3986:            (double)INT_MAX ||
3987:            OFLOW_MAX - *lengthptr < delta)
3988:        {
3989:          *errorcodeptr = ERR20;
3990:          goto FAILED;
3991:        }
3992:        *lengthptr += delta;
3993:      }
3994:
3995:      /* This is compiling for real */
3996:
3997:    else
3998:    {
3999:      if (groupsetfirstbyte && reqbyte < 0) reqbyte = firstbyte;
4000:      for (i = 1; i < repeat_min; i++)
4001:      {
4002:        uschar *hc;
4003:        uschar *this_hwm = cd->hwm;
4004:        memcpy(code, previous, len);
4005:        for (hc = save_hwm; hc < this_hwm; hc += LINK_SIZE)
4006:        {

```

```

4007:         PUT(cd->hwm, 0, GET(hc, 0) + len);
4008:         cd->hwm += LINK_SIZE;
4009:     }
4010:     save_hwm = this_hwm;
4011:     code += len;
4012: }
4013: }
4014: }
4015:
4016: if (repeat_max > 0) repeat_max -= repeat_min;
4017: }
4018:
4019: /* This code is common to both the zero and non-zero minimum cases. If
4020: the maximum is limited, it replicates the group in a nested fashion,
4021: remembering the bracket starts on a stack. In the case of a zero minimum,
4022: the first one was set up above. In all cases the repeat_max now specifies
4023: the number of additional copies needed. Again, we must remember to
4024: replicate entries on the forward reference list. */
4025:
4026: if (repeat_max >= 0)
4027: {
4028:     /* In the pre-compile phase, we don't actually do the replication. We
4029: just adjust the length as if we had. For each repetition we must add 1
4030: to the length for BRAZERO and for all but the last repetition we must
4031: add 2 + 2*LINKSIZE to allow for the nesting that occurs. Do some
4032: paranoid checks to avoid integer overflow. */
4033:
4034: if (lengthptr != NULL && repeat_max > 0)
4035: {
4036:     int delta = repeat_max * (length_prevgroup + 1 + 2 + 2*LINK_SIZE) -
4037:         2 - 2*LINK_SIZE; /* Last one doesn't nest */
4038:     if (((double)repeat_max *
4039:         (double)(length_prevgroup + 1 + 2 + 2*LINK_SIZE)
4040:         > (double)INT_MAX ||
4041:         OFLOW_MAX - *lengthptr < delta)
4042:     {
4043:         *errorcodeptr = ERR20;
4044:         goto FAILED;
4045:     }
4046:     *lengthptr += delta;
4047: }

```

```

4048:
4049:     /* This is compiling for real */
4050:
4051:     else for (i = repeat_max - 1; i >= 0; i--)
4052:     {
4053:         uschar *hc;
4054:         uschar *this_hwm = cd->hwm;
4055:
4056:         *code++ = OP_BRAZERO + repeat_type;
4057:
4058:         /* All but the final copy start a new nesting, maintaining the
4059:         chain of brackets outstanding. */
4060:
4061:         if (i != 0)
4062:         {
4063:             int offset;
4064:             *code++ = OP_BRA;
4065:             offset = (bralink == NULL)? 0 : code - bralink;
4066:             bralink = code;
4067:             PUTINC(code, 0, offset);
4068:         }
4069:
4070:         memcpy(code, previous, len);
4071:         for (hc = save_hwm; hc < this_hwm; hc += LINK_SIZE)
4072:         {
4073:             PUT(cd->hwm, 0, GET(hc, 0) + len + ((i != 0)? 2+LINK_SIZE : 1));
4074:             cd->hwm += LINK_SIZE;
4075:         }
4076:         save_hwm = this_hwm;
4077:         code += len;
4078:     }
4079:
4080:     /* Now chain through the pending brackets, and fill in their length
4081:     fields (which are holding the chain links pro tem). */
4082:
4083:     while (bralink != NULL)
4084:     {
4085:         int oldlinkoffset;
4086:         int offset = code - bralink + 1;
4087:         uschar *bra = code - offset;
4088:         oldlinkoffset = GET(bra, 1);

```

```

4089:     bralink = (oldlinkoffset == 0)? NULL : bralink - oldlinkoffset;
4090:     *code++ = OP_KET;
4091:     PUTINC(code, 0, offset);
4092:     PUT(bra, 1, offset);
4093: }
4094: }
4095:
4096: /* If the maximum is unlimited, set a repeater in the final copy. We
4097: can't just offset backwards from the current code point, because we
4098: don't know if there's been an options resetting after the ket. The
4099: correct offset was computed above.
4100:
4101: Then, when we are doing the actual compile phase, check to see whether
4102: this group is a non-atomic one that could match an empty string. If so,
4103: convert the initial operator to the S form (e.g. OP_BRA -> OP_SBRA) so
4104: that runtime checking can be done. [This check is also applied to
4105: atomic groups at runtime, but in a different way.] */
4106:
4107: else
4108: {
4109:     uschar *ketcode = code - ketoffset;
4110:     uschar *bracode = ketcode - GET(ketcode, 1);
4111:     *ketcode = OP_KETRMX + repeat_type;
4112:     if (lengthptr == NULL && *bracode != OP_ONCE)
4113:     {
4114:         uschar *scode = bracode;
4115:         do
4116:         {
4117:             if (could_be_empty_branch(scode, ketcode, utf8))
4118:             {
4119:                 *bracode += OP_SBRA - OP_BRA;
4120:                 break;
4121:             }
4122:             scode += GET(scode, 1);
4123:         }
4124:         while (*scode == OP_ALT);
4125:     }
4126: }
4127: }
4128:
4129: /* If previous is OP_FAIL, it was generated by an empty class [] in

```

```

4130: JavaScript mode. The other ways in which OP_FAIL can be generated, that is
4131: by (*FAIL) or (!) set previous to NULL, which gives a "nothing to repeat"
4132: error above. We can just ignore the repeat in JS case. */
4133:
4134: else if (*previous == OP_FAIL) goto END_REPEAT;
4135:
4136: /* Else there's some kind of shambles */
4137:
4138: else
4139: {
4140:     *errorcodeptr = ERR11;
4141:     goto FAILED;
4142: }
4143:
4144: /* If the character following a repeat is '+', or if certain optimization
4145: tests above succeeded, possessive_quantifier is TRUE. For some of the
4146: simpler opcodes, there is an special alternative opcode for this. For
4147: anything else, we wrap the entire repeated item inside OP_ONCE brackets.
4148: The '+' notation is just syntactic sugar, taken from Sun's Java package,
4149: but the special opcodes can optimize it a bit. The repeated item starts at
4150: tempcode, not at previous, which might be the first part of a string whose
4151: (former) last char we repeated.
4152:
4153: Possessifying an 'exact' quantifier has no effect, so we can ignore it. But
4154: an 'upto' may follow. We skip over an 'exact' item, and then test the
4155: length of what remains before proceeding. */
4156:
4157: if (possessive_quantifier)
4158: {
4159:     int len;
4160:     if (*tempcode == OP_EXACT || *tempcode == OP_TYPEEXACT ||
4161:         *tempcode == OP_NOTEXACT)
4162:         tempcode += _pcre_OP_lengths[*tempcode] +
4163:             ((*tempcode == OP_TYPEEXACT &&
4164:              (tempcode[3] == OP_PROP || tempcode[3] == OP_NOTPROP))? 2:0);
4165:     len = code - tempcode;
4166:     if (len > 0) switch (*tempcode)
4167:     {
4168:         case OP_STAR: *tempcode = OP_POSSTAR; break;
4169:         case OP_PLUS: *tempcode = OP_POSPLUS; break;
4170:         case OP_QUERY: *tempcode = OP_POSQUERY; break;

```



```

4171:     case OP_UPTO: *tempcode = OP_POSUPTO; break;
4172:
4173:     case OP_TYPESTAR: *tempcode = OP_TYPEPOSSTAR; break;
4174:     case OP_TYPEPLUS: *tempcode = OP_TYPEPOSPLUS; break;
4175:     case OP_TYPEQUERY: *tempcode = OP_TYPEPOSQUERY; break;
4176:     case OP_TYPEUPTO: *tempcode = OP_TYPEPOSUPTO; break;
4177:
4178:     case OP_NOTSTAR: *tempcode = OP_NOTPOSSTAR; break;
4179:     case OP_NOTPLUS: *tempcode = OP_NOTPOSPLUS; break;
4180:     case OP_NOTQUERY: *tempcode = OP_NOTPOSQUERY; break;
4181:     case OP_NOTUPTO: *tempcode = OP_NOTPOSUPTO; break;
4182:
4183:     default:
4184:     memmove(tempcode + 1+LINK_SIZE, tempcode, len);
4185:     code += 1 + LINK_SIZE;
4186:     len += 1 + LINK_SIZE;
4187:     tempcode[0] = OP_ONCE;
4188:     *code++ = OP_KET;
4189:     PUTINC(code, 0, len);
4190:     PUT(tempcode, 1, len);
4191:     break;
4192: }
4193: }
4194:
4195: /* In all case we no longer have a previous item. We also set the
4196: "follows varying string" flag for subsequently encountered reqbytes if
4197: it isn't already set and we have just passed a varying length item. */
4198:
4199: END_REPEAT:
4200: previous = NULL;
4201: cd->req_varyopt |= reqvary;
4202: break;
4203:
4204:
4205:
=====
4206: /* Start of nested parenthesized sub-expression, or comment or lookahead or
4207: lookbehind or option setting or condition or all the other extended
4208: parenthesis forms. */
4209:
4210: case '(':

```

```

4211: newoptions = options;
4212: skipbytes = 0;
4213: bravalue = OP_CBRA;
4214: save_hwm = cd->hwm;
4215: reset_bracount = FALSE;
4216:
4217: /* First deal with various "verbs" that can be introduced by '*'. */
4218:
4219: if (*(++ptr) == '*' && (cd->ctypes[ptr[1]] & ctype_letter) != 0)
4220: {
4221:     int i, namelen;
4222:     const char *vn = verbnames;
4223:     const uschar *name = ++ptr;
4224:     previous = NULL;
4225:     while ((cd->ctypes[*++ptr] & ctype_letter) != 0) {};
4226:     if (*ptr == ':')
4227:     {
4228:         *errorcodeptr = ERR59; /* Not supported */
4229:         goto FAILED;
4230:     }
4231:     if (*ptr != ')')
4232:     {
4233:         *errorcodeptr = ERR60;
4234:         goto FAILED;
4235:     }
4236:     namelen = ptr - name;
4237:     for (i = 0; i < verbcount; i++)
4238:     {
4239:         if (namelen == verbs[i].len &&
4240:             strncmp((char *)name, vn, namelen) == 0)
4241:         {
4242:             *code = verbs[i].op;
4243:             if (*code++ == OP_ACCEPT) cd->had_accept = TRUE;
4244:             break;
4245:         }
4246:         vn += verbs[i].len + 1;
4247:     }
4248:     if (i < verbcount) continue;
4249:     *errorcodeptr = ERR60;
4250:     goto FAILED;
4251: }

```

```

4252:
4253: /* Deal with the extended parentheses; all are introduced by '?', and the
4254: appearance of any of them means that this is not a capturing group. */
4255:
4256: else if (*ptr == '?')
4257: {
4258:     int i, set, unset, namelen;
4259:     int *optset;
4260:     const uschar *name;
4261:     uschar *slot;
4262:
4263:     switch (*(++ptr))
4264:     {
4265:         case '#':          /* Comment; skip to ket */
4266:             ptr++;
4267:             while (*ptr != 0 && *ptr != ')') ptr++;
4268:             if (*ptr == 0)
4269:             {
4270:                 *errorcodeptr = ERR18;
4271:                 goto FAILED;
4272:             }
4273:             continue;
4274:
4275:
4276:         /* ----- */
4277:         case '|':          /* Reset capture count for each branch */
4278:             reset_bracount = TRUE;
4279:             /* Fall through */
4280:
4281:         /* ----- */
4282:         case ':':          /* Non-capturing bracket */
4283:             bravalue = OP_BRA;
4284:             ptr++;
4285:             break;
4286:
4287:
4288:         /* ----- */
4289:         case '(':
4290:             bravalue = OP_COND;    /* Conditional group */
4291:
4292:             /* A condition can be an assertion, a number (referring to a numbered

```

```

4293:    group), a name (referring to a named group), or 'R', referring to
4294:    recursion. R<digits> and R&name are also permitted for recursion tests.
4295:
4296:    There are several syntaxes for testing a named group: (?<name>) is used
4297:    by Python; Perl 5.10 onwards uses (?(<name>)) or (?('name')).
4298:
4299:    There are two unfortunate ambiguities, caused by history. (a) 'R' can
4300:    be the recursive thing or the name 'R' (and similarly for 'R' followed
4301:    by digits), and (b) a number could be a name that consists of digits.
4302:    In both cases, we look for a name first; if not found, we try the other
4303:    cases. */
4304:
4305:    /* For conditions that are assertions, check the syntax, and then exit
4306:    the switch. This will take control down to where bracketed groups,
4307:    including assertions, are processed. */
4308:
4309:    if (ptr[1] == '?' && (ptr[2] == '=' || ptr[2] == '!' || ptr[2] == '<'))
4310:        break;
4311:
4312:    /* Most other conditions use OP_CREF (a couple change to OP_RREF
4313:    below), and all need to skip 3 bytes at the start of the group. */
4314:
4315:    code[1+LINK_SIZE] = OP_CREF;
4316:    skipbytes = 3;
4317:    refsign = -1;
4318:
4319:    /* Check for a test for recursion in a named group. */
4320:
4321:    if (ptr[1] == 'R' && ptr[2] == '&')
4322:    {
4323:        terminator = -1;
4324:        ptr += 2;
4325:        code[1+LINK_SIZE] = OP_RREF; /* Change the type of test */
4326:    }
4327:
4328:    /* Check for a test for a named group's having been set, using the Perl
4329:    syntax (?(<name>)) or (?('name')) */
4330:
4331:    else if (ptr[1] == '<')
4332:    {
4333:        terminator = '>';

```

```

4334:     ptr++;
4335: }
4336: else if (ptr[1] == '\\")
4337: {
4338:     terminator = '\\";
4339:     ptr++;
4340: }
4341: else
4342: {
4343:     terminator = 0;
4344:     if (ptr[1] == '-' || ptr[1] == '+') refsign = *(++ptr);
4345: }
4346:
4347: /* We now expect to read a name; any thing else is an error */
4348:
4349: if ((cd->ctypes[ptr[1]] & ctype_word) == 0)
4350: {
4351:     ptr += 1; /* To get the right offset */
4352:     *errorcodeptr = ERR28;
4353:     goto FAILED;
4354: }
4355:
4356: /* Read the name, but also get it as a number if it's all digits */
4357:
4358: recno = 0;
4359: name = ++ptr;
4360: while ((cd->ctypes[*ptr] & ctype_word) != 0)
4361: {
4362:     if (recno >= 0)
4363:         recno = ((digitab[*ptr] & ctype_digit) != 0)?
4364:             recno * 10 + *ptr - '0' : -1;
4365:     ptr++;
4366: }
4367: namelen = ptr - name;
4368:
4369: if ((terminator > 0 && *ptr++ != terminator) || *ptr++ != ')')
4370: {
4371:     ptr--; /* Error offset */
4372:     *errorcodeptr = ERR26;
4373:     goto FAILED;
4374: }

```

```

4375:
4376: /* Do no further checking in the pre-compile phase. */
4377:
4378: if (lengthptr != NULL) break;
4379:
4380: /* In the real compile we do the work of looking for the actual
4381: reference. If the string started with "+" or "-" we require the rest to
4382: be digits, in which case recno will be set. */
4383:
4384: if (refsign > 0)
4385: {
4386:     if (recno <= 0)
4387:     {
4388:         *errorcodeptr = ERR58;
4389:         goto FAILED;
4390:     }
4391:     recno = (refsign == '-')?
4392:         cd->bracount - recno + 1 : recno + cd->bracount;
4393:     if (recno <= 0 || recno > cd->final_bracount)
4394:     {
4395:         *errorcodeptr = ERR15;
4396:         goto FAILED;
4397:     }
4398:     PUT2(code, 2+LINK_SIZE, recno);
4399:     break;
4400: }
4401:
4402: /* Otherwise (did not start with "+" or "-"), start by looking for the
4403: name. */
4404:
4405: slot = cd->name_table;
4406: for (i = 0; i < cd->names_found; i++)
4407: {
4408:     if (strncmp((char *)name, (char *)slot+2, namelen) == 0) break;
4409:     slot += cd->name_entry_size;
4410: }
4411:
4412: /* Found a previous named subpattern */
4413:
4414: if (i < cd->names_found)
4415: {

```

```

4416:     recno = GET2(slot, 0);
4417:     PUT2(code, 2+LINK_SIZE, recno);
4418: }
4419:
4420: /* Search the pattern for a forward reference */
4421:
4422: else if ((i = find_parens(ptr, cd, name, namelen,
4423:     (options & PCRE_EXTENDED) != 0)) > 0)
4424: {
4425:     PUT2(code, 2+LINK_SIZE, i);
4426: }
4427:
4428: /* If terminator == 0 it means that the name followed directly after
4429: the opening parenthesis [e.g. (?abc)...] and in this case there are
4430: some further alternatives to try. For the cases where terminator != 0
4431: [things like (?(<name>... or (?('name')... or (?(&name)... ] we have
4432: now checked all the possibilities, so give an error. */
4433:
4434: else if (terminator != 0)
4435: {
4436:     *errorcodeptr = ERR15;
4437:     goto FAILED;
4438: }
4439:
4440: /* Check for (?R) for recursion. Allow digits after R to specify a
4441: specific group number. */
4442:
4443: else if (*name == 'R')
4444: {
4445:     recno = 0;
4446:     for (i = 1; i < namelen; i++)
4447:     {
4448:         if ((digitab[name[i]] & ctype_digit) == 0)
4449:         {
4450:             *errorcodeptr = ERR15;
4451:             goto FAILED;
4452:         }
4453:         recno = recno * 10 + name[i] - '0';
4454:     }
4455:     if (recno == 0) recno = RREF_ANY;
4456:     code[1+LINK_SIZE] = OP_RREF;    /* Change test type */

```

```

4457:     PUT2(code, 2+LINK_SIZE, recno);
4458: }
4459:
4460: /* Similarly, check for the (? (DEFINE) "condition", which is always
4461: false. */
4462:
4463: else if (namelen == 6 && strncmp((char *)name, "DEFINE", 6) == 0)
4464: {
4465:     code[1+LINK_SIZE] = OP_DEF;
4466:     skipbytes = 1;
4467: }
4468:
4469: /* Check for the "name" actually being a subpattern number. We are
4470: in the second pass here, so final_bracount is set. */
4471:
4472: else if (recno > 0 && recno <= cd->final_bracount)
4473: {
4474:     PUT2(code, 2+LINK_SIZE, recno);
4475: }
4476:
4477: /* Either an unidentified subpattern, or a reference to (? (0) */
4478:
4479: else
4480: {
4481:     *errorcodeptr = (recno == 0)? ERR35: ERR15;
4482:     goto FAILED;
4483: }
4484: break;
4485:
4486:
4487: /* ----- */
4488: case '=':          /* Positive lookahead */
4489:     bravalue = OP_ASSERT;
4490:     ptr++;
4491:     break;
4492:
4493:
4494: /* ----- */
4495: case '!':          /* Negative lookahead */
4496:     ptr++;
4497:     if (*ptr == ')')    /* Optimize (!) */

```



```

4498:    {
4499:        *code++ = OP_FAIL;
4500:        previous = NULL;
4501:        continue;
4502:    }
4503:    bravalue = OP_ASSERT_NOT;
4504:    break;
4505:
4506:
4507:    /* ----- */
4508:    case '<':        /* Lookbehind or named define */
4509:    switch (ptr[1])
4510:    {
4511:        case '=':    /* Positive lookbehind */
4512:            bravalue = OP_ASSERTBACK;
4513:            ptr += 2;
4514:            break;
4515:
4516:        case '!':    /* Negative lookbehind */
4517:            bravalue = OP_ASSERTBACK_NOT;
4518:            ptr += 2;
4519:            break;
4520:
4521:        default:    /* Could be name define, else bad */
4522:            if ((cd->ctypes[ptr[1]] & ctype_word) != 0) goto DEFINE_NAME;
4523:            ptr++;    /* Correct offset for error */
4524:            *errorcodeptr = ERR24;
4525:            goto FAILED;
4526:        }
4527:    break;
4528:
4529:
4530:    /* ----- */
4531:    case '>':        /* One-time brackets */
4532:        bravalue = OP_ONCE;
4533:        ptr++;
4534:        break;
4535:
4536:
4537:    /* ----- */
4538:    case 'C':        /* Callout - may be followed by digits; */

```

```

4539: previous_callout = code; /* Save for later completion */
4540: after_manual_callout = 1; /* Skip one item before completing */
4541: *code++ = OP_CALLOUT;
4542: {
4543:     int n = 0;
4544:     while ((digitab[*(++ptr)] & ctype_digit) != 0)
4545:         n = n * 10 + *ptr - '0';
4546:     if (*ptr != ')')
4547:     {
4548:         *errorcodeptr = ERR39;
4549:         goto FAILED;
4550:     }
4551:     if (n > 255)
4552:     {
4553:         *errorcodeptr = ERR38;
4554:         goto FAILED;
4555:     }
4556:     *code++ = n;
4557:     PUT(code, 0, ptr - cd->start_pattern + 1); /* Pattern offset */
4558:     PUT(code, LINK_SIZE, 0); /* Default length */
4559:     code += 2 * LINK_SIZE;
4560: }
4561: previous = NULL;
4562: continue;
4563:
4564:
4565: /* ----- */
4566: case 'P': /* Python-style named subpattern handling */
4567: if (*(++ptr) == '=' || *ptr == '>') /* Reference or recursion */
4568: {
4569:     is_recurse = *ptr == '>';
4570:     terminator = ')';
4571:     goto NAMED_REF_OR_RECURSE;
4572: }
4573: else if (*ptr != '<') /* Test for Python-style definition */
4574: {
4575:     *errorcodeptr = ERR41;
4576:     goto FAILED;
4577: }
4578: /* Fall through to handle (?P< as (?< is handled */
4579:

```

```

4580:
4581: /* ----- */
4582: DEFINE_NAME: /* Come here from (?< handling */
4583: case '\":
4584: {
4585:     terminator = (*ptr == '<')? '>' : '\\";
4586:     name = ++ptr;
4587:
4588:     while ((cd->ctypes[*ptr] & ctype_word) != 0) ptr++;
4589:     namelen = ptr - name;
4590:
4591:     /* In the pre-compile phase, just do a syntax check. */
4592:
4593:     if (lengthptr != NULL)
4594:     {
4595:         if (*ptr != terminator)
4596:         {
4597:             *errorcodeptr = ERR42;
4598:             goto FAILED;
4599:         }
4600:         if (cd->names_found >= MAX_NAME_COUNT)
4601:         {
4602:             *errorcodeptr = ERR49;
4603:             goto FAILED;
4604:         }
4605:         if (namelen + 3 > cd->name_entry_size)
4606:         {
4607:             cd->name_entry_size = namelen + 3;
4608:             if (namelen > MAX_NAME_SIZE)
4609:             {
4610:                 *errorcodeptr = ERR48;
4611:                 goto FAILED;
4612:             }
4613:         }
4614:     }
4615:
4616:     /* In the real compile, create the entry in the table */
4617:
4618:     else
4619:     {
4620:         slot = cd->name_table;

```

```

4621:     for (i = 0; i < cd->names_found; i++)
4622:     {
4623:         int crc = memcmp(name, slot+2, namelen);
4624:         if (crc == 0)
4625:         {
4626:             if (slot[2+namelen] == 0)
4627:             {
4628:                 if ((options & PCRE_DUPNAMES) == 0)
4629:                 {
4630:                     *errorcodeptr = ERR43;
4631:                     goto FAILED;
4632:                 }
4633:             }
4634:             else crc = -1;    /* Current name is substring */
4635:         }
4636:         if (crc < 0)
4637:         {
4638:             memmove(slot + cd->name_entry_size, slot,
4639:                 (cd->names_found - i) * cd->name_entry_size);
4640:             break;
4641:         }
4642:         slot += cd->name_entry_size;
4643:     }
4644:
4645:     PUT2(slot, 0, cd->bracount + 1);
4646:     memcpy(slot + 2, name, namelen);
4647:     slot[2+namelen] = 0;
4648: }
4649: }
4650:
4651: /* In both cases, count the number of names we've encountered. */
4652:
4653: ptr++;          /* Move past > or ' */
4654: cd->names_found++;
4655: goto NUMBERED_GROUP;
4656:
4657:
4658: /* ----- */
4659: case '&':        /* Perl recursion/subroutine syntax */
4660:     terminator = ')';
4661:     is_recurse = TRUE;

```

```

4662:  /* Fall through */
4663:
4664:  /* We come here from the Python syntax above that handles both
4665:  references (?P=name) and recursion (?P>name), as well as falling
4666:  through from the Perl recursion syntax (?&name). We also come here from
4667:  the Perl \k<name> or \k'name' back reference syntax and the \k{name}
4668:  .NET syntax, and the Oniguruma \g<...> and \g'...' subroutine syntax. */
4669:
4670:  NAMED_REF_OR_RECURSE:
4671:  name = ++ptr;
4672:  while ((cd->ctypes[*ptr] & ctype_word) != 0) ptr++;
4673:  namelen = ptr - name;
4674:
4675:  /* In the pre-compile phase, do a syntax check and set a dummy
4676:  reference number. */
4677:
4678:  if (lengthptr != NULL)
4679:  {
4680:  if (namelen == 0)
4681:  {
4682:  *errorcodeptr = ERR62;
4683:  goto FAILED;
4684:  }
4685:  if (*ptr != terminator)
4686:  {
4687:  *errorcodeptr = ERR42;
4688:  goto FAILED;
4689:  }
4690:  if (namelen > MAX_NAME_SIZE)
4691:  {
4692:  *errorcodeptr = ERR48;
4693:  goto FAILED;
4694:  }
4695:  recno = 0;
4696:  }
4697:
4698:  /* In the real compile, seek the name in the table. We check the name
4699:  first, and then check that we have reached the end of the name in the
4700:  table. That way, if the name that is longer than any in the table,
4701:  the comparison will fail without reading beyond the table entry. */
4702:

```

```

4703:     else
4704:     {
4705:         slot = cd->name_table;
4706:         for (i = 0; i < cd->names_found; i++)
4707:         {
4708:             if (strncmp((char *)name, (char *)slot+2, namelen) == 0 &&
4709:                 slot[2+namelen] == 0)
4710:                 break;
4711:             slot += cd->name_entry_size;
4712:         }
4713:
4714:         if (i < cd->names_found)      /* Back reference */
4715:         {
4716:             recno = GET2(slot, 0);
4717:         }
4718:         else if ((recno =             /* Forward back reference */
4719:             find_parens(ptr, cd, name, namelen,
4720:                 (options & PCRE_EXTENDED) != 0)) <= 0)
4721:         {
4722:             *errorcodeptr = ERR15;
4723:             goto FAILED;
4724:         }
4725:     }
4726:
4727:     /* In both phases, we can now go to the code than handles numerical
4728:     recursion or backreferences. */
4729:
4730:     if (is_recurse) goto HANDLE_RECURSION;
4731:     else goto HANDLE_REFERENCE;
4732:
4733:
4734:     /* ----- */
4735:     case 'R':          /* Recursion */
4736:     ptr++;             /* Same as (?0) */
4737:     /* Fall through */
4738:
4739:
4740:     /* ----- */
4741:     case '-': case '+':
4742:     case '0': case '1': case '2': case '3': case '4': /* Recursion or */
4743:     case '5': case '6': case '7': case '8': case '9': /* subroutine */

```

```

4744:     {
4745:     const uschar *called;
4746:     terminator = ')';
4747:
4748:     /* Come here from the \g<...> and \g'...' code (Oniguruma
4749:     compatibility). However, the syntax has been checked to ensure that
4750:     the ... are a (signed) number, so that neither ERR63 nor ERR29 will
4751:     be called on this path, nor with the jump to OTHER_CHAR_AFTER_QUERY
4752:     ever be taken. */
4753:
4754:     HANDLE_NUMERICAL_RECURSION:
4755:
4756:     if ((refsign = *ptr) == '+')
4757:     {
4758:         ptr++;
4759:         if ((digitab[*ptr] & ctype_digit) == 0)
4760:         {
4761:             *errorcodeptr = ERR63;
4762:             goto FAILED;
4763:         }
4764:     }
4765:     else if (refsign == '-')
4766:     {
4767:         if ((digitab[ptr[1]] & ctype_digit) == 0)
4768:             goto OTHER_CHAR_AFTER_QUERY;
4769:         ptr++;
4770:     }
4771:
4772:     recno = 0;
4773:     while((digitab[*ptr] & ctype_digit) != 0)
4774:         recno = recno * 10 + *ptr++ - '0';
4775:
4776:     if (*ptr != terminator)
4777:     {
4778:         *errorcodeptr = ERR29;
4779:         goto FAILED;
4780:     }
4781:
4782:     if (refsign == '-')
4783:     {
4784:         if (recno == 0)

```

```

4785:     {
4786:         *errorcodeptr = ERR58;
4787:         goto FAILED;
4788:     }
4789:     recno = cd->bracount - recno + 1;
4790:     if (recno <= 0)
4791:     {
4792:         *errorcodeptr = ERR15;
4793:         goto FAILED;
4794:     }
4795: }
4796: else if (refsign == '+')
4797: {
4798:     if (recno == 0)
4799:     {
4800:         *errorcodeptr = ERR58;
4801:         goto FAILED;
4802:     }
4803:     recno += cd->bracount;
4804: }
4805:
4806: /* Come here from code above that handles a named recursion */
4807:
4808: HANDLE_RECURSION:
4809:
4810: previous = code;
4811: called = cd->start_code;
4812:
4813: /* When we are actually compiling, find the bracket that is being
4814: referenced. Temporarily end the regex in case it doesn't exist before
4815: this point. If we end up with a forward reference, first check that
4816: the bracket does occur later so we can give the error (and position)
4817: now. Then remember this forward reference in the workspace so it can
4818: be filled in at the end. */
4819:
4820: if (lengthptr == NULL)
4821: {
4822:     *code = OP_END;
4823:     if (recno != 0) called = find_bracket(cd->start_code, utf8, recno);
4824:
4825:     /* Forward reference */

```



```

4826:
4827:     if (called == NULL)
4828:     {
4829:         if (find_parens(ptr, cd, NULL, recno,
4830:             (options & PCRE_EXTENDED) != 0) < 0)
4831:         {
4832:             *errorcodeptr = ERR15;
4833:             goto FAILED;
4834:         }
4835:         called = cd->start_code + recno;
4836:         PUTINC(cd->hwm, 0, code + 2 + LINK_SIZE - cd->start_code);
4837:     }
4838:
4839:     /* If not a forward reference, and the subpattern is still open,
4840:     this is a recursive call. We check to see if this is a left
4841:     recursion that could loop for ever, and diagnose that case. */
4842:
4843:     else if (GET(called, 1) == 0 &&
4844:         could_be_empty(called, code, bcptr, utf8))
4845:     {
4846:         *errorcodeptr = ERR40;
4847:         goto FAILED;
4848:     }
4849: }
4850:
4851: /* Insert the recursion/subroutine item, automatically wrapped inside
4852: "once" brackets. Set up a "previous group" length so that a
4853: subsequent quantifier will work. */
4854:
4855: *code = OP_ONCE;
4856: PUT(code, 1, 2 + 2*LINK_SIZE);
4857: code += 1 + LINK_SIZE;
4858:
4859: *code = OP_RECURSE;
4860: PUT(code, 1, called - cd->start_code);
4861: code += 1 + LINK_SIZE;
4862:
4863: *code = OP_KET;
4864: PUT(code, 1, 2 + 2*LINK_SIZE);
4865: code += 1 + LINK_SIZE;
4866:

```

```

4867:     length_prevgroup = 3 + 3*LINK_SIZE;
4868: }
4869:
4870: /* Can't determine a first byte now */
4871:
4872: if (firstbyte == REQ_UNSET) firstbyte = REQ_NONE;
4873: continue;
4874:
4875:
4876: /* ----- */
4877: default: /* Other characters: check option setting */
4878: OTHER_CHAR_AFTER_QUERY:
4879:     set = unset = 0;
4880:     optset = &set;
4881:
4882:     while (*ptr != ')' && *ptr != ':')
4883:     {
4884:         switch (*ptr++)
4885:         {
4886:             case '-': optset = &unset; break;
4887:
4888:             case 'J': /* Record that it changed in the external options */
4889:                 *optset |= PCRE_DUPNAMES;
4890:                 cd->external_flags |= PCRE_JCHANGED;
4891:                 break;
4892:
4893:             case 'i': *optset |= PCRE_CASELESS; break;
4894:             case 'm': *optset |= PCRE_MULTILINE; break;
4895:             case 's': *optset |= PCRE_DOTALL; break;
4896:             case 'x': *optset |= PCRE_EXTENDED; break;
4897:             case 'U': *optset |= PCRE_UNGREEDY; break;
4898:             case 'X': *optset |= PCRE_EXTRA; break;
4899:
4900:             default: *errorcodeptr = ERR12;
4901:                 ptr--; /* Correct the offset */
4902:                 goto FAILED;
4903:         }
4904:     }
4905:
4906: /* Set up the changed option bits, but don't change anything yet. */
4907:

```

```

4908:     newoptions = (options | set) & (~unset);
4909:
4910:     /* If the options ended with ')' this is not the start of a nested
4911:     group with option changes, so the options change at this level. If this
4912:     item is right at the start of the pattern, the options can be
4913:     abstracted and made external in the pre-compile phase, and ignored in
4914:     the compile phase. This can be helpful when matching -- for instance in
4915:     caseless checking of required bytes.
4916:
4917:     If the code pointer is not (cd->start_code + 1 + LINK_SIZE), we are
4918:     definitely *not* at the start of the pattern because something has been
4919:     compiled. In the pre-compile phase, however, the code pointer can have
4920:     that value after the start, because it gets reset as code is discarded
4921:     during the pre-compile. However, this can happen only at top level - if
4922:     we are within parentheses, the starting BRA will still be present. At
4923:     any parenthesis level, the length value can be used to test if anything
4924:     has been compiled at that level. Thus, a test for both these conditions
4925:     is necessary to ensure we correctly detect the start of the pattern in
4926:     both phases.
4927:
4928:     If we are not at the pattern start, compile code to change the ims
4929:     options if this setting actually changes any of them, and reset the
4930:     greedy defaults and the case value for firstbyte and reqbyte. */
4931:
4932:     if (*ptr == ')')
4933:     {
4934:         if (code == cd->start_code + 1 + LINK_SIZE &&
4935:             (lengthptr == NULL || *lengthptr == 2 + 2*LINK_SIZE))
4936:         {
4937:             cd->external_options = newoptions;
4938:         }
4939:     else
4940:     {
4941:         if ((options & PCRE_IMS) != (newoptions & PCRE_IMS))
4942:         {
4943:             *code++ = OP_OPT;
4944:             *code++ = newoptions & PCRE_IMS;
4945:         }
4946:         greedy_default = ((newoptions & PCRE_UNGREEDY) != 0);
4947:         greedy_non_default = greedy_default ^ 1;
4948:         req_caseopt = ((newoptions & PCRE_CASELESS) != 0)? REQ_CASELESS : 0;

```

```

4949:     }
4950:
4951:     /* Change options at this level, and pass them back for use
4952:     in subsequent branches. When not at the start of the pattern, this
4953:     information is also necessary so that a resetting item can be
4954:     compiled at the end of a group (if we are in a group). */
4955:
4956:     *optionsptr = options = newoptions;
4957:     previous = NULL;    /* This item can't be repeated */
4958:     continue;          /* It is complete */
4959: }
4960:
4961: /* If the options ended with ':' we are heading into a nested group
4962: with possible change of options. Such groups are non-capturing and are
4963: not assertions of any kind. All we need to do is skip over the ':';
4964: the newoptions value is handled below. */
4965:
4966: bravalue = OP_BRA;
4967: ptr++;
4968: } /* End of switch for character following (? */
4969: } /* End of (? handling */
4970:
4971: /* Opening parenthesis not followed by '?'. If PCRE_NO_AUTO_CAPTURE is set,
4972: all unadorned brackets become non-capturing and behave like (?:... )
4973: brackets. */
4974:
4975: else if ((options & PCRE_NO_AUTO_CAPTURE) != 0)
4976: {
4977:     bravalue = OP_BRA;
4978: }
4979:
4980: /* Else we have a capturing group. */
4981:
4982: else
4983: {
4984:     NUMBERED_GROUP:
4985:     cd->bracount += 1;
4986:     PUT2(code, 1+LINK_SIZE, cd->bracount);
4987:     skipbytes = 2;
4988: }
4989:

```

```

4990:  /* Process nested bracketed regex. Assertions may not be repeated, but
4991:  other kinds can be. All their opcodes are >= OP_ONCE. We copy code into a
4992:  non-register variable in order to be able to pass its address because some
4993:  compilers complain otherwise. Pass in a new setting for the ims options if
4994:  they have changed. */
4995:
4996:  previous = (bravalue >= OP_ONCE)? code : NULL;
4997:  *code = bravalue;
4998:  tempcode = code;
4999:  tempreqvary = cd->req_varyopt;  /* Save value before bracket */
5000:  length_prevgroup = 0;          /* Initialize for pre-compile phase */
5001:
5002:  if (!compile_regex(
5003:      newoptions,                /* The complete new option state */
5004:      options & PCRE_IMS,       /* The previous ims option state */
5005:      &tempcode,                /* Where to put code (updated) */
5006:      &ptr,                     /* Input pointer (updated) */
5007:      errorcodeptr,             /* Where to put an error message */
5008:      (bravalue == OP_ASSERTBACK ||
5009:       bravalue == OP_ASSERTBACK_NOT), /* TRUE if back assert */
5010:      reset_bracount,           /* True if (?| group */
5011:      skipbytes,                /* Skip over bracket number */
5012:      &subfirstbyte,            /* For possible first char */
5013:      &subreqbyte,              /* For possible last char */
5014:      bcptr,                    /* Current branch chain */
5015:      cd,                       /* Tables block */
5016:      (lengthptr == NULL)? NULL : /* Actual compile phase */
5017:      &length_prevgroup         /* Pre-compile phase */
5018:  ))
5019:      goto FAILED;
5020:
5021:  /* At the end of compiling, code is still pointing to the start of the
5022:  group, while tempcode has been updated to point past the end of the group
5023:  and any option resetting that may follow it. The pattern pointer (ptr)
5024:  is on the bracket. */
5025:
5026:  /* If this is a conditional bracket, check that there are no more than
5027:  two branches in the group, or just one if it's a DEFINE group. We do this
5028:  in the real compile phase, not in the pre-pass, where the whole group may
5029:  not be available. */
5030:

```

```

5031: if (bravalue == OP_COND && lengthptr == NULL)
5032: {
5033:     uschar *tc = code;
5034:     int condcount = 0;
5035:
5036:     do {
5037:         condcount++;
5038:         tc += GET(tc,1);
5039:     }
5040:     while (*tc != OP_KET);
5041:
5042:     /* A DEFINE group is never obeyed inline (the "condition" is always
5043:     false). It must have only one branch. */
5044:
5045:     if (code[LINK_SIZE+1] == OP_DEF)
5046:     {
5047:         if (condcount > 1)
5048:         {
5049:             *errorcodeptr = ERR54;
5050:             goto FAILED;
5051:         }
5052:         bravalue = OP_DEF; /* Just a flag to suppress char handling below */
5053:     }
5054:
5055:     /* A "normal" conditional group. If there is just one branch, we must not
5056:     make use of its firstbyte or reqbyte, because this is equivalent to an
5057:     empty second branch. */
5058:
5059:     else
5060:     {
5061:         if (condcount > 2)
5062:         {
5063:             *errorcodeptr = ERR27;
5064:             goto FAILED;
5065:         }
5066:         if (condcount == 1) subfirstbyte = subreqbyte = REQ_NONE;
5067:     }
5068: }
5069:
5070: /* Error if hit end of pattern */
5071:

```

```

5072:  if (*ptr != ')')
5073:  {
5074:      *errorcodeptr = ERR14;
5075:      goto FAILED;
5076:  }
5077:
5078:  /* In the pre-compile phase, update the length by the length of the group,
5079:  less the brackets at either end. Then reduce the compiled code to just a
5080:  set of non-capturing brackets so that it doesn't use much memory if it is
5081:  duplicated by a quantifier.*/
5082:
5083:  if (lengthptr != NULL)
5084:  {
5085:      if (OFLOW_MAX - *lengthptr < length_prevgroup - 2 - 2*LINK_SIZE)
5086:      {
5087:          *errorcodeptr = ERR20;
5088:          goto FAILED;
5089:      }
5090:      *lengthptr += length_prevgroup - 2 - 2*LINK_SIZE;
5091:      *code++ = OP_BRA;
5092:      PUTINC(code, 0, 1 + LINK_SIZE);
5093:      *code++ = OP_KET;
5094:      PUTINC(code, 0, 1 + LINK_SIZE);
5095:      break;  /* No need to waste time with special character handling */
5096:  }
5097:
5098:  /* Otherwise update the main code pointer to the end of the group. */
5099:
5100:  code = tempcode;
5101:
5102:  /* For a DEFINE group, required and first character settings are not
5103:  relevant. */
5104:
5105:  if (bravalue == OP_DEF) break;
5106:
5107:  /* Handle updating of the required and first characters for other types of
5108:  group. Update for normal brackets of all kinds, and conditions with two
5109:  branches (see code above). If the bracket is followed by a quantifier with
5110:  zero repeat, we have to back off. Hence the definition of zeroreqbyte and
5111:  zerofirstbyte outside the main loop so that they can be accessed for the
5112:  back off. */

```

```

5113:
5114:  zeroreqbyte = reqbyte;
5115:  zerofirstbyte = firstbyte;
5116:  groupsetfirstbyte = FALSE;
5117:
5118:  if (bravalue >= OP_ONCE)
5119:  {
5120:    /* If we have not yet set a firstbyte in this branch, take it from the
5121:       subpattern, remembering that it was set here so that a repeat of more
5122:       than one can replicate it as reqbyte if necessary. If the subpattern has
5123:       no firstbyte, set "none" for the whole branch. In both cases, a zero
5124:       repeat forces firstbyte to "none". */
5125:
5126:    if (firstbyte == REQ_UNSET)
5127:    {
5128:      if (subfirstbyte >= 0)
5129:      {
5130:        firstbyte = subfirstbyte;
5131:        groupsetfirstbyte = TRUE;
5132:      }
5133:      else firstbyte = REQ_NONE;
5134:      zerofirstbyte = REQ_NONE;
5135:    }
5136:
5137:    /* If firstbyte was previously set, convert the subpattern's firstbyte
5138:       into reqbyte if there wasn't one, using the vary flag that was in
5139:       existence beforehand. */
5140:
5141:    else if (subfirstbyte >= 0 && subreqbyte < 0)
5142:      subreqbyte = subfirstbyte | tempreqvary;
5143:
5144:    /* If the subpattern set a required byte (or set a first byte that isn't
5145:       really the first byte - see above), set it. */
5146:
5147:    if (subreqbyte >= 0) reqbyte = subreqbyte;
5148:  }
5149:
5150:  /* For a forward assertion, we take the reqbyte, if set. This can be
5151:     helpful if the pattern that follows the assertion doesn't set a different
5152:     char. For example, it's useful for /(=?abcde).+/. We can't set firstbyte
5153:     for an assertion, however because it leads to incorrect effect for patterns

```



```

5154: such as /(?=a)a.+ / when the "real" "a" would then become a reqbyte instead
5155: of a firstbyte. This is overcome by a scan at the end if there's no
5156: firstbyte, looking for an asserted first char. */
5157:
5158: else if (bravalue == OP_ASSERT && subreqbyte >= 0) reqbyte = subreqbyte;
5159: break; /* End of processing '(' */
5160:
5161:
5162: ===== */
5163: /* Handle metasequences introduced by \. For ones like \d, the ESC_ values
5164: are arranged to be the negation of the corresponding OP_values. For the
5165: back references, the values are ESC_REF plus the reference number. Only
5166: back references and those types that consume a character may be repeated.
5167: We can test for values between ESC_b and ESC_Z for the latter; this may
5168: have to change if any new ones are ever created. */
5169:
5170: case '\\':
5171: tempptr = ptr;
5172: c = check_escape(&ptr, errorcodeptr, cd->bracount, options, FALSE);
5173: if (*errorcodeptr != 0) goto FAILED;
5174:
5175: if (c < 0)
5176: {
5177: if (-c == ESC_Q) /* Handle start of quoted string */
5178: {
5179: if (ptr[1] == '\\' && ptr[2] == 'E') ptr += 2; /* avoid empty string */
5180: else inescq = TRUE;
5181: continue;
5182: }
5183:
5184: if (-c == ESC_E) continue; /* Perl ignores an orphan \E */
5185:
5186: /* For metasequences that actually match a character, we disable the
5187: setting of a first character if it hasn't already been set. */
5188:
5189: if (firstbyte == REQ_UNSET && -c > ESC_b && -c < ESC_Z)
5190: firstbyte = REQ_NONE;
5191:
5192: /* Set values to reset to if this is followed by a zero repeat. */
5193:

```

```

5194: zerofirstbyte = firstbyte;
5195: zeroreqbyte = reqbyte;
5196:
5197: /* \g<name> or \g'name' is a subroutine call by name and \g<n> or \g'n'
5198: is a subroutine call by number (Oniguruma syntax). In fact, the value
5199: -ESC_g is returned only for these cases. So we don't need to check for <
5200: or ' if the value is -ESC_g. For the Perl syntax \g{n} the value is
5201: -ESC_REF+n, and for the Perl syntax \g{name} the result is -ESC_k (as
5202: that is a synonym for a named back reference). */
5203:
5204: if (-c == ESC_g)
5205: {
5206:     const uschar *p;
5207:     save_hwm = cd->hwm; /* Normally this is set when '(' is read */
5208:     terminator = (*(++ptr) == '<')? '>' : '\';
5209:
5210:     /* These two statements stop the compiler for warning about possibly
5211:     unset variables caused by the jump to HANDLE_NUMERICAL_RECURSION. In
5212:     fact, because we actually check for a number below, the paths that
5213:     would actually be in error are never taken. */
5214:
5215:     skipbytes = 0;
5216:     reset_bracount = FALSE;
5217:
5218:     /* Test for a name */
5219:
5220:     if (ptr[1] != '+' && ptr[1] != '-')
5221:     {
5222:         BOOL isnumber = TRUE;
5223:         for (p = ptr + 1; *p != 0 && *p != terminator; p++)
5224:         {
5225:             if ((cd->ctypes[*p] & ctype_digit) == 0) isnumber = FALSE;
5226:             if ((cd->ctypes[*p] & ctype_word) == 0) break;
5227:         }
5228:         if (*p != terminator)
5229:         {
5230:             *errorcodeptr = ERR57;
5231:             break;
5232:         }
5233:         if (isnumber)
5234:         {

```

```

5235:     ptr++;
5236:     goto HANDLE_NUMERICAL_RECURSION;
5237: }
5238: is_recurse = TRUE;
5239: goto NAMED_REF_OR_RECURSE;
5240: }
5241:
5242: /* Test a signed number in angle brackets or quotes. */
5243:
5244: p = ptr + 2;
5245: while ((digitab[*p] & ctype_digit) != 0) p++;
5246: if (*p != terminator)
5247: {
5248:     *errorcodeptr = ERR57;
5249:     break;
5250: }
5251: ptr++;
5252: goto HANDLE_NUMERICAL_RECURSION;
5253: }
5254:
5255: /* \k<name> or \k'name' is a back reference by name (Perl syntax).
5256: We also support \k{name} (.NET syntax) */
5257:
5258: if (-c == ESC_k && (ptr[1] == '<' || ptr[1] == '"' || ptr[1] == '{'))
5259: {
5260:     is_recurse = FALSE;
5261:     terminator = (*(++ptr) == '<')? '>' : (*ptr == '"")? '"' : '}';
5262:     goto NAMED_REF_OR_RECURSE;
5263: }
5264:
5265: /* Back references are handled specially; must disable firstbyte if
5266: not set to cope with cases like (?=(\w+)) \1: which would otherwise set
5267: ':' later. */
5268:
5269: if (-c >= ESC_REF)
5270: {
5271:     recno = -c - ESC_REF;
5272:
5273:     HANDLE_REFERENCE: /* Come here from named backref handling */
5274:     if (firstbyte == REQ_UNSET) firstbyte = REQ_NONE;
5275:     previous = code;

```

```

5276:     *code++ = OP_REF;
5277:     PUT2INC(code, 0, recno);
5278:     cd->backref_map |= (recno < 32)? (1 << recno) : 1;
5279:     if (recno > cd->top_backref) cd->top_backref = recno;
5280: }
5281:
5282: /* So are Unicode property matches, if supported. */
5283:
5284: #ifdef SUPPORT_UCP
5285:     else if (-c == ESC_P || -c == ESC_p)
5286:     {
5287:         BOOL negated;
5288:         int pdata;
5289:         int ptype = get_ucp(&ptr, &negated, &pdata, errorcodeptr);
5290:         if (ptype < 0) goto FAILED;
5291:         previous = code;
5292:         *code++ = ((-c == ESC_p) != negated)? OP_PROP : OP_NOTPROP;
5293:         *code++ = ptype;
5294:         *code++ = pdata;
5295:     }
5296: #else
5297:
5298: /* If Unicode properties are not supported, \X, \P, and \p are not
5299: allowed. */
5300:
5301:     else if (-c == ESC_X || -c == ESC_P || -c == ESC_p)
5302:     {
5303:         *errorcodeptr = ERR45;
5304:         goto FAILED;
5305:     }
5306: #endif
5307:
5308: /* For the rest (including \X when Unicode properties are supported), we
5309: can obtain the OP value by negating the escape value. */
5310:
5311:     else
5312:     {
5313:         previous = (-c > ESC_b && -c < ESC_Z)? code : NULL;
5314:         *code++ = -c;
5315:     }
5316:     continue;

```

```

5317:   }
5318:
5319:   /* We have a data character whose value is in c. In UTF-8 mode it may have
5320:   a value > 127. We set its representation in the length/buffer, and then
5321:   handle it as a data character. */
5322:
5323: #ifdef SUPPORT_UTF8
5324:   if (utf8 && c > 127)
5325:     mclength = _pcre_ord2utf8(c, mcbuffer);
5326:   else
5327: #endif
5328:
5329:   {
5330:     mcbuffer[0] = c;
5331:     mclength = 1;
5332:   }
5333:   goto ONE_CHAR;
5334:
5335:
5336: =====*/
5337:   /* Handle a literal character. It is guaranteed not to be whitespace or #
5338:   when the extended flag is set. If we are in UTF-8 mode, it may be a
5339:   multi-byte literal character. */
5340:
5341:   default:
5342:     NORMAL_CHAR:
5343:     mclength = 1;
5344:     mcbuffer[0] = c;
5345:
5346: #ifdef SUPPORT_UTF8
5347:     if (utf8 && c >= 0xc0)
5348:     {
5349:       while ((ptr[1] & 0xc0) == 0x80)
5350:         mcbuffer[mclength++] = *(++ptr);
5351:     }
5352: #endif
5353:
5354:   /* At this point we have the character's bytes in mcbuffer, and the length
5355:   in mclength. When not in UTF-8 mode, the length is always 1. */
5356:

```

```

5357: ONE_CHAR:
5358: previous = code;
5359: *code++ = ((options & PCRE_CASELESS) != 0)? OP_CHARNC : OP_CHAR;
5360: for (c = 0; c < mclength; c++) *code++ = mcbuffer[c];
5361:
5362: /* Remember if \r or \n were seen */
5363:
5364: if (mcbuffer[0] == '\r' || mcbuffer[0] == '\n')
5365:     cd->external_flags |= PCRE_HASCORLRF;
5366:
5367: /* Set the first and required bytes appropriately. If no previous first
5368: byte, set it from this character, but revert to none on a zero repeat.
5369: Otherwise, leave the firstbyte value alone, and don't change it on a zero
5370: repeat. */
5371:
5372: if (firstbyte == REQ_UNSET)
5373: {
5374:     zerofirstbyte = REQ_NONE;
5375:     zeroreqbyte = reqbyte;
5376:
5377: /* If the character is more than one byte long, we can set firstbyte
5378: only if it is not to be matched caselessly. */
5379:
5380: if (mclength == 1 || req_caseopt == 0)
5381: {
5382:     firstbyte = mcbuffer[0] | req_caseopt;
5383:     if (mclength != 1) reqbyte = code[-1] | cd->req_varyopt;
5384: }
5385: else firstbyte = reqbyte = REQ_NONE;
5386: }
5387:
5388: /* firstbyte was previously set; we can set reqbyte only the length is
5389: 1 or the matching is caseful. */
5390:
5391: else
5392: {
5393:     zerofirstbyte = firstbyte;
5394:     zeroreqbyte = reqbyte;
5395:     if (mclength == 1 || req_caseopt == 0)
5396:         reqbyte = code[-1] | req_caseopt | cd->req_varyopt;
5397: }

```

```

5398:
5399:  break;          /* End of literal character handling */
5400:  }
5401:  }              /* end of big loop */
5402:
5403:
5404: /* Control never reaches here by falling through, only by a goto for all the
5405: error states. Pass back the position in the pattern so that it can be displayed
5406: to the user for diagnosing the error. */
5407:
5408: FAILED:
5409: *ptrptr = ptr;
5410: return FALSE;
5411: }
5412:
5413:
5414:
5415:
5416: /*****
5417: *   Compile sequence of alternatives      *
5418: *****/
5419:
5420: /* On entry, ptr is pointing past the bracket character, but on return it
5421: points to the closing bracket, or vertical bar, or end of string. The code
5422: variable is pointing at the byte into which the BRA operator has been stored.
5423: If the ims options are changed at the start (for a (?ims: group) or during any
5424: branch, we need to insert an OP_OPT item at the start of every following branch
5425: to ensure they get set correctly at run time, and also pass the new options
5426: into every subsequent branch compile.
5427:
5428: This function is used during the pre-compile phase when we are trying to find
5429: out the amount of memory needed, as well as during the real compile phase. The
5430: value of lengthptr distinguishes the two phases.
5431:
5432: Arguments:
5433:  options    option bits, including any changes for this subpattern
5434:  oldims     previous settings of ims option bits
5435:  codeptr    -> the address of the current code pointer
5436:  ptrptr     -> the address of the current pattern pointer
5437:  errorcodeptr -> pointer to error code variable
5438:  lookbehind TRUE if this is a lookbehind assertion

```

```

5439: reset_bracount TRUE to reset the count for each branch
5440: skipbytes    skip this many bytes at start (for brackets and OP_COND)
5441: firstbyteptr  place to put the first required character, or a negative number
5442: reqbyteptr    place to put the last required character, or a negative number
5443: bcptr         pointer to the chain of currently open branches
5444: cd            points to the data block with tables pointers etc.
5445: lengthptr     NULL during the real compile phase
5446:              points to length accumulator during pre-compile phase
5447:
5448: Returns:      TRUE on success
5449: */
5450:
5451: static BOOL
5452: compile_regex(int options, int oldims, uschar **codeptr, const uschar **ptrptr,
5453: int *errorcodeptr, BOOL lookbehind, BOOL reset_bracount, int skipbytes,
5454: int *firstbyteptr, int *reqbyteptr, branch_chain *bcptr, compile_data *cd,
5455: int *lengthptr)
5456: {
5457: const uschar *ptr = *ptrptr;
5458: uschar *code = *codeptr;
5459: uschar *last_branch = code;
5460: uschar *start_bracket = code;
5461: uschar *reverse_count = NULL;
5462: int firstbyte, reqbyte;
5463: int branchfirstbyte, branchreqbyte;
5464: int length;
5465: int orig_bracount;
5466: int max_bracount;
5467: branch_chain bc;
5468:
5469: bc.outer = bcptr;
5470: bc.current = code;
5471:
5472: firstbyte = reqbyte = REQ_UNSET;
5473:
5474: /* Accumulate the length for use in the pre-compile phase. Start with the
5475: length of the BRA and KET and any extra bytes that are required at the
5476: beginning. We accumulate in a local variable to save frequent testing of
5477: lenthptr for NULL. We cannot do this by looking at the value of code at the
5478: start and end of each alternative, because compiled items are discarded during
5479: the pre-compile phase so that the work space is not exceeded. */

```



```

5480:
5481: length = 2 + 2*LINK_SIZE + skipbytes;
5482:
5483: /* WARNING: If the above line is changed for any reason, you must also change
5484: the code that abstracts option settings at the start of the pattern and makes
5485: them global. It tests the value of length for (2 + 2*LINK_SIZE) in the
5486: pre-compile phase to find out whether anything has yet been compiled or not. */
5487:
5488: /* Offset is set zero to mark that this bracket is still open */
5489:
5490: PUT(code, 1, 0);
5491: code += 1 + LINK_SIZE + skipbytes;
5492:
5493: /* Loop for each alternative branch */
5494:
5495: orig_bracount = max_bracount = cd->bracount;
5496: for (;;)
5497: {
5498: /* For a (?| group, reset the capturing bracket count so that each branch
5499: uses the same numbers. */
5500:
5501: if (reset_bracount) cd->bracount = orig_bracount;
5502:
5503: /* Handle a change of ims options at the start of the branch */
5504:
5505: if ((options & PCRE_IMS) != oldims)
5506: {
5507: *code++ = OP_OPT;
5508: *code++ = options & PCRE_IMS;
5509: length += 2;
5510: }
5511:
5512: /* Set up dummy OP_REVERSE if lookbehind assertion */
5513:
5514: if (lookbehind)
5515: {
5516: *code++ = OP_REVERSE;
5517: reverse_count = code;
5518: PUTINC(code, 0, 0);
5519: length += 1 + LINK_SIZE;
5520: }

```

```

5521:
5522: /* Now compile the branch; in the pre-compile phase its length gets added
5523: into the length. */
5524:
5525: if (!compile_branch(&options, &code, &ptr, errorcodeptr, &branchfirstbyte,
5526:     &branchreqbyte, &bc, cd, (lengthptr == NULL)? NULL : &length))
5527: {
5528:     *ptrptr = ptr;
5529:     return FALSE;
5530: }
5531:
5532: /* Keep the highest bracket count in case (?| was used and some branch
5533: has fewer than the rest. */
5534:
5535: if (cd->bracount > max_bracount) max_bracount = cd->bracount;
5536:
5537: /* In the real compile phase, there is some post-processing to be done. */
5538:
5539: if (lengthptr == NULL)
5540: {
5541:     /* If this is the first branch, the firstbyte and reqbyte values for the
5542:     branch become the values for the regex. */
5543:
5544:     if (*last_branch != OP_ALT)
5545:     {
5546:         firstbyte = branchfirstbyte;
5547:         reqbyte = branchreqbyte;
5548:     }
5549:
5550:     /* If this is not the first branch, the first char and reqbyte have to
5551:     match the values from all the previous branches, except that if the
5552:     previous value for reqbyte didn't have REQ_VARY set, it can still match,
5553:     and we set REQ_VARY for the regex. */
5554:
5555:     else
5556:     {
5557:         /* If we previously had a firstbyte, but it doesn't match the new branch,
5558:         we have to abandon the firstbyte for the regex, but if there was
5559:         previously no reqbyte, it takes on the value of the old firstbyte. */
5560:
5561:         if (firstbyte >= 0 && firstbyte != branchfirstbyte)

```

```

5562:    {
5563:    if (reqbyte < 0) reqbyte = firstbyte;
5564:    firstbyte = REQ_NONE;
5565:    }
5566:
5567:    /* If we (now or from before) have no firstbyte, a firstbyte from the
5568:    branch becomes a reqbyte if there isn't a branch reqbyte. */
5569:
5570:    if (firstbyte < 0 && branchfirstbyte >= 0 && branchreqbyte < 0)
5571:        branchreqbyte = branchfirstbyte;
5572:
5573:    /* Now ensure that the reqbytes match */
5574:
5575:    if ((reqbyte & ~REQ_VARY) != (branchreqbyte & ~REQ_VARY))
5576:        reqbyte = REQ_NONE;
5577:    else reqbyte |= branchreqbyte; /* To "or" REQ_VARY */
5578:    }
5579:
5580:    /* If lookbehind, check that this branch matches a fixed-length string, and
5581:    put the length into the OP_REVERSE item. Temporarily mark the end of the
5582:    branch with OP_END. */
5583:
5584:    if (lookbehind)
5585:    {
5586:        int fixed_length;
5587:        *code = OP_END;
5588:        fixed_length = find_fixedlength(last_branch, options);
5589:        DPRINTF(("fixed length = %d \n", fixed_length));
5590:        if (fixed_length < 0)
5591:        {
5592:            *errorcodeptr = (fixed_length == -2)? ERR36 : ERR25;
5593:            *ptrptr = ptr;
5594:            return FALSE;
5595:        }
5596:        PUT(reverse_count, 0, fixed_length);
5597:    }
5598:    }
5599:
5600:    /* Reached end of expression, either ')' or end of pattern. In the real
5601:    compile phase, go back through the alternative branches and reverse the chain
5602:    of offsets, with the field in the BRA item now becoming an offset to the

```

```

5603: first alternative. If there are no alternatives, it points to the end of the
5604: group. The length in the terminating ket is always the length of the whole
5605: bracketed item. If any of the ims options were changed inside the group,
5606: compile a resetting op-code following, except at the very end of the pattern.
5607: Return leaving the pointer at the terminating char. */
5608:
5609: if (*ptr != '|')
5610: {
5611:     if (lengthptr == NULL)
5612:     {
5613:         int branch_length = code - last_branch;
5614:         do
5615:         {
5616:             int prev_length = GET(last_branch, 1);
5617:             PUT(last_branch, 1, branch_length);
5618:             branch_length = prev_length;
5619:             last_branch -= branch_length;
5620:         }
5621:         while (branch_length > 0);
5622:     }
5623:
5624:     /* Fill in the ket */
5625:
5626:     *code = OP_KET;
5627:     PUT(code, 1, code - start_bracket);
5628:     code += 1 + LINK_SIZE;
5629:
5630:     /* Resetting option if needed */
5631:
5632:     if ((options & PCRE_IMS) != oldims && *ptr == '|')
5633:     {
5634:         *code++ = OP_OPT;
5635:         *code++ = oldims;
5636:         length += 2;
5637:     }
5638:
5639:     /* Retain the highest bracket number, in case resetting was used. */
5640:
5641:     cd->bracount = max_bracount;
5642:
5643:     /* Set values to pass back */

```

```

5644:
5645:  *codeptr = code;
5646:  *ptrptr = ptr;
5647:  *firstbyteptr = firstbyte;
5648:  *reqbyteptr = reqbyte;
5649:  if (lengthptr != NULL)
5650:  {
5651:      if (OFLOW_MAX - *lengthptr < length)
5652:      {
5653:          *errorcodeptr = ERR20;
5654:          return FALSE;
5655:      }
5656:      *lengthptr += length;
5657:  }
5658:  return TRUE;
5659:  }
5660:
5661:  /* Another branch follows. In the pre-compile phase, we can move the code
5662:  pointer back to where it was for the start of the first branch. (That is,
5663:  pretend that each branch is the only one.)
5664:
5665:  In the real compile phase, insert an ALT node. Its length field points back
5666:  to the previous branch while the bracket remains open. At the end the chain
5667:  is reversed. It's done like this so that the start of the bracket has a
5668:  zero offset until it is closed, making it possible to detect recursion. */
5669:
5670:  if (lengthptr != NULL)
5671:  {
5672:      code = *codeptr + 1 + LINK_SIZE + skipbytes;
5673:      length += 1 + LINK_SIZE;
5674:  }
5675:  else
5676:  {
5677:      *code = OP_ALT;
5678:      PUT(code, 1, code - last_branch);
5679:      bc.current = last_branch = code;
5680:      code += 1 + LINK_SIZE;
5681:  }
5682:
5683:  ptr++;
5684:  }

```

```

5685: /* Control never reaches here */
5686: }
5687:
5688:
5689:
5690:
5691: /*****
5692: *      Check for anchored expression      *
5693: *****/
5694:
5695: /* Try to find out if this is an anchored regular expression. Consider each
5696: alternative branch. If they all start with OP_SOD or OP_CIRC, or with a bracket
5697: all of whose alternatives start with OP_SOD or OP_CIRC (recurse ad lib), then
5698: it's anchored. However, if this is a multiline pattern, then only OP_SOD
5699: counts, since OP_CIRC can match in the middle.
5700:
5701: We can also consider a regex to be anchored if OP_SOM starts all its branches.
5702: This is the code for \G, which means "match at start of match position, taking
5703: into account the match offset".
5704:
5705: A branch is also implicitly anchored if it starts with .* and DOTALL is set,
5706: because that will try the rest of the pattern at all possible matching points,
5707: so there is no point trying again.... er ....
5708:
5709: .... except when the .* appears inside capturing parentheses, and there is a
5710: subsequent back reference to those parentheses. We haven't enough information
5711: to catch that case precisely.
5712:
5713: At first, the best we could do was to detect when .* was in capturing brackets
5714: and the highest back reference was greater than or equal to that level.
5715: However, by keeping a bitmap of the first 31 back references, we can catch some
5716: of the more common cases more precisely.
5717:
5718: Arguments:
5719: code      points to start of expression (the bracket)
5720: options   points to the options setting
5721: bracket_map a bitmap of which brackets we are inside while testing; this
5722:             handles up to substring 31; after that we just have to take
5723:             the less precise approach
5724: backref_map the back reference bitmap
5725:

```

```

5726: Returns:   TRUE or FALSE
5727: */
5728:
5729: static BOOL
5730: is_anchored(register const uschar *code, int *options, unsigned int bracket_map,
5731: unsigned int backref_map)
5732: {
5733: do {
5734:   const uschar *scode = first_significant_code(code + _pcre_OP_lengths[*code],
5735:   options, PCRE_MULTILINE, FALSE);
5736:   register int op = *scode;
5737:
5738:   /* Non-capturing brackets */
5739:
5740:   if (op == OP_BRA)
5741:   {
5742:     if (!is_anchored(scode, options, bracket_map, backref_map)) return FALSE;
5743:   }
5744:
5745:   /* Capturing brackets */
5746:
5747:   else if (op == OP_CBRA)
5748:   {
5749:     int n = GET2(scode, 1+LINK_SIZE);
5750:     int new_map = bracket_map | ((n < 32)? (1 << n) : 1);
5751:     if (!is_anchored(scode, options, new_map, backref_map)) return FALSE;
5752:   }
5753:
5754:   /* Other brackets */
5755:
5756:   else if (op == OP_ASSERT || op == OP_ONCE || op == OP_COND)
5757:   {
5758:     if (!is_anchored(scode, options, bracket_map, backref_map)) return FALSE;
5759:   }
5760:
5761:   /* .* is not anchored unless DOTALL is set (which generates OP_ALLANY) and
5762:   it isn't in brackets that are or may be referenced. */
5763:
5764:   else if ((op == OP_TYPESTAR || op == OP_TYPEMINSTAR ||
5765:   op == OP_TYPEPOSSTAR))
5766:   {

```

```

5767:   if (scode[1] != OP_ALLANY || (bracket_map & backref_map) != 0)
5768:       return FALSE;
5769:   }
5770:
5771:   /* Check for explicit anchoring */
5772:
5773:   else if (op != OP_SOD && op != OP_SOM &&
5774:           ((*options & PCRE_MULTILINE) != 0 || op != OP_CIRC))
5775:       return FALSE;
5776:   code += GET(code, 1);
5777:   }
5778: while (*code == OP_ALT); /* Loop for each alternative */
5779: return TRUE;
5780: }
5781:
5782:
5783:
5784: /*****
5785:  *      Check for starting with ^ or .*      *
5786:  *****/
5787:
5788: /* This is called to find out if every branch starts with ^ or .* so that
5789: "first char" processing can be done to speed things up in multiline
5790: matching and for non-DOTALL patterns that start with .* (which must start at
5791: the beginning or after \n). As in the case of is_anchored() (see above), we
5792: have to take account of back references to capturing brackets that contain .*
5793: because in that case we can't make the assumption.
5794:
5795: Arguments:
5796: code      points to start of expression (the bracket)
5797: bracket_map  a bitmap of which brackets we are inside while testing; this
5798:             handles up to substring 31; after that we just have to take
5799:             the less precise approach
5800: backref_map  the back reference bitmap
5801:
5802: Returns:    TRUE or FALSE
5803: */
5804:
5805: static BOOL
5806: is_startline(const uschar *code, unsigned int bracket_map,
5807: unsigned int backref_map)

```



```

5808: {
5809: do {
5810:     const uschar *scode = first_significant_code(code + _pcre_OP_lengths[*code],
5811:         NULL, 0, FALSE);
5812:     register int op = *scode;
5813:
5814:     /* Non-capturing brackets */
5815:
5816:     if (op == OP_BRA)
5817:     {
5818:         if (!is_startline(scode, bracket_map, backref_map)) return FALSE;
5819:     }
5820:
5821:     /* Capturing brackets */
5822:
5823:     else if (op == OP_CBRA)
5824:     {
5825:         int n = GET2(scode, 1+LINK_SIZE);
5826:         int new_map = bracket_map | ((n < 32)? (1 << n) : 1);
5827:         if (!is_startline(scode, new_map, backref_map)) return FALSE;
5828:     }
5829:
5830:     /* Other brackets */
5831:
5832:     else if (op == OP_ASSERT || op == OP_ONCE || op == OP_COND)
5833:     { if (!is_startline(scode, bracket_map, backref_map)) return FALSE; }
5834:
5835:     /* .* means "start at start or after \n" if it isn't in brackets that
5836:     may be referenced. */
5837:
5838:     else if (op == OP_TYPESTAR || op == OP_TYPEMINSTAR || op == OP_TYPEPOSSTAR)
5839:     {
5840:         if (scode[1] != OP_ANY || (bracket_map & backref_map) != 0) return FALSE;
5841:     }
5842:
5843:     /* Check for explicit circumflex */
5844:
5845:     else if (op != OP_CIRC) return FALSE;
5846:
5847:     /* Move on to the next alternative */
5848:

```

```

5849: code += GET(code, 1);
5850: }
5851: while (*code == OP_ALT); /* Loop for each alternative */
5852: return TRUE;
5853: }
5854:
5855:
5856:
5857: /*****
5858:  *    Check for asserted fixed first char    *
5859: *****/
5860:
5861: /* During compilation, the "first char" settings from forward assertions are
5862: discarded, because they can cause conflicts with actual literals that follow.
5863: However, if we end up without a first char setting for an unanchored pattern,
5864: it is worth scanning the regex to see if there is an initial asserted first
5865: char. If all branches start with the same asserted char, or with a bracket all
5866: of whose alternatives start with the same asserted char (recurse ad lib), then
5867: we return that char, otherwise -1.
5868:
5869: Arguments:
5870: code    points to start of expression (the bracket)
5871: options  pointer to the options (used to check casing changes)
5872: inassert TRUE if in an assertion
5873:
5874: Returns:  -1 or the fixed first char
5875: */
5876:
5877: static int
5878: find_firstassertedchar(const uschar *code, int *options, BOOL inassert)
5879: {
5880: register int c = -1;
5881: do {
5882:     int d;
5883:     const uschar *scode =
5884:         first_significant_code(code + 1+LINK_SIZE, options, PCRE_CASELESS, TRUE);
5885:     register int op = *scode;
5886:
5887:     switch(op)
5888:     {
5889:     default:

```

```

5890:   return -1;
5891:
5892:   case OP_BRA:
5893:   case OP_CBRA:
5894:   case OP_ASSERT:
5895:   case OP_ONCE:
5896:   case OP_COND:
5897:   if ((d = find_firstassertedchar(scode, options, op == OP_ASSERT)) < 0)
5898:       return -1;
5899:   if (c < 0) c = d; else if (c != d) return -1;
5900:   break;
5901:
5902:   case OP_EXACT:      /* Fall through */
5903:   scode += 2;
5904:
5905:   case OP_CHAR:
5906:   case OP_CHARNC:
5907:   case OP_PLUS:
5908:   case OP_MINPLUS:
5909:   case OP_POSPLUS:
5910:   if (!inassert) return -1;
5911:   if (c < 0)
5912:   {
5913:       c = scode[1];
5914:       if ((*options & PCRE_CASELESS) != 0) c |= REQ_CASELESS;
5915:   }
5916:   else if (c != scode[1]) return -1;
5917:   break;
5918:   }
5919:
5920:   code += GET(code, 1);
5921:   }
5922: while (*code == OP_ALT);
5923: return c;
5924: }
5925:
5926:
5927:
5928: /*****
5929:  *      Compile a Regular Expression      *
5930:  *****/

```

```

5931:
5932: /* This function takes a string and returns a pointer to a block of store
5933: holding a compiled version of the expression. The original API for this
5934: function had no error code return variable; it is retained for backwards
5935: compatibility. The new function is given a new name.
5936:
5937: Arguments:
5938: pattern    the regular expression
5939: options    various option bits
5940: errorcodeptr pointer to error code variable (pcre_compile2() only)
5941:           can be NULL if you don't want a code value
5942: errorptr    pointer to pointer to error text
5943: erroroffset ptr offset in pattern where error was detected
5944: tables      pointer to character tables or NULL
5945:
5946: Returns:    pointer to compiled data block, or NULL on error,
5947:           with errorptr and erroroffset set
5948: */
5949:
5950: PCRE_EXP_DEFN pcre * PCRE_CALL_CONVENTION
5951: pcre_compile(const char *pattern, int options, const char **errorptr,
5952: int *erroroffset, const unsigned char *tables)
5953: {
5954: return pcre_compile2(pattern, options, NULL, errorptr, erroroffset, tables);
5955: }
5956:
5957:
5958: PCRE_EXP_DEFN pcre * PCRE_CALL_CONVENTION
5959: pcre_compile2(const char *pattern, int options, int *errorcodeptr,
5960: const char **errorptr, int *erroroffset, const unsigned char *tables)
5961: {
5962: real_pcre *re;
5963: int length = 1; /* For final END opcode */
5964: int firstbyte, reqbyte, newline;
5965: int errorcode = 0;
5966: int skipatstart = 0;
5967: #ifdef SUPPORT_UTF8
5968: BOOL utf8;
5969: #endif
5970: size_t size;
5971: uschar *code;

```

```

5972: const uschar *codestart;
5973: const uschar *ptr;
5974: compile_data compile_block;
5975: compile_data *cd = &compile_block;
5976:
5977: /* This space is used for "compiling" into during the first phase, when we are
5978: computing the amount of memory that is needed. Compiled items are thrown away
5979: as soon as possible, so that a fairly large buffer should be sufficient for
5980: this purpose. The same space is used in the second phase for remembering where
5981: to fill in forward references to subpatterns. */
5982:
5983: uschar cworkspace[COMPILE_WORK_SIZE];
5984:
5985: /* Set this early so that early errors get offset 0. */
5986:
5987: ptr = (const uschar *)pattern;
5988:
5989: /* We can't pass back an error message if errorptr is NULL; I guess the best we
5990: can do is just return NULL, but we can set a code value if there is a code
5991: pointer. */
5992:
5993: if (errorptr == NULL)
5994: {
5995:   if (errorcodeptr != NULL) *errorcodeptr = 99;
5996:   return NULL;
5997: }
5998:
5999: *errorptr = NULL;
6000: if (errorcodeptr != NULL) *errorcodeptr = ERR0;
6001:
6002: /* However, we can give a message for this error */
6003:
6004: if (erroroffset == NULL)
6005: {
6006:   errorcode = ERR16;
6007:   goto PCRE_EARLY_ERROR_RETURN2;
6008: }
6009:
6010: *erroroffset = 0;
6011:
6012: /* Can't support UTF8 unless PCRE has been compiled to include the code. */

```

```

6013:
6014: #ifdef SUPPORT_UTF8
6015: utf8 = (options & PCRE_UTF8) != 0;
6016: if (utf8 && (options & PCRE_NO_UTF8_CHECK) == 0 &&
6017:     (*erroroffset = _pcre_valid_utf8((uschar *)pattern, -1)) >= 0)
6018: {
6019:     errorcode = ERR44;
6020:     goto PCRE_EARLY_ERROR_RETURN2;
6021: }
6022: #else
6023: if ((options & PCRE_UTF8) != 0)
6024: {
6025:     errorcode = ERR32;
6026:     goto PCRE_EARLY_ERROR_RETURN;
6027: }
6028: #endif
6029:
6030: if ((options & ~PUBLIC_OPTIONS) != 0)
6031: {
6032:     errorcode = ERR17;
6033:     goto PCRE_EARLY_ERROR_RETURN;
6034: }
6035:
6036: /* Set up pointers to the individual character tables */
6037:
6038: if (tables == NULL) tables = _pcre_default_tables;
6039: cd->lcc = tables + lcc_offset;
6040: cd->fcc = tables + fcc_offset;
6041: cd->cbits = tables + cbits_offset;
6042: cd->ctypes = tables + ctypes_offset;
6043:
6044: /* Check for global one-time settings at the start of the pattern, and remember
6045: the offset for later. */
6046:
6047: while (ptr[skipatstart] == '(' && ptr[skipatstart+1] == '*')
6048: {
6049:     int newnl = 0;
6050:     int newbsr = 0;
6051:
6052:     if (strncmp((char *) (ptr+skipatstart+2), "CR"), 3) == 0)
6053:         { skipatstart += 5; newnl = PCRE_NEWLINE_CR; }

```

```

6054: else if (strncmp((char *) (ptr+skipatstart+2), "LF"), 3) == 0)
6055:     { skipatstart += 5; newnl = PCRE_NEWLINE_LF; }
6056: else if (strncmp((char *) (ptr+skipatstart+2), "CRLF"), 5) == 0)
6057:     { skipatstart += 7; newnl = PCRE_NEWLINE_CR + PCRE_NEWLINE_LF; }
6058: else if (strncmp((char *) (ptr+skipatstart+2), "ANY"), 4) == 0)
6059:     { skipatstart += 6; newnl = PCRE_NEWLINE_ANY; }
6060: else if (strncmp((char *) (ptr+skipatstart+2), "ANYCRLF"), 8) == 0)
6061:     { skipatstart += 10; newnl = PCRE_NEWLINE_ANYCRLF; }
6062:
6063: else if (strncmp((char *) (ptr+skipatstart+2), "BSR_ANYCRLF"), 12) == 0)
6064:     { skipatstart += 14; newbsr = PCRE_BSR_ANYCRLF; }
6065: else if (strncmp((char *) (ptr+skipatstart+2), "BSR_UNICODE"), 12) == 0)
6066:     { skipatstart += 14; newbsr = PCRE_BSR_UNICODE; }
6067:
6068: if (newnl != 0)
6069:     options = (options & ~PCRE_NEWLINE_BITS) | newnl;
6070: else if (newbsr != 0)
6071:     options = (options & ~(PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE)) | newbsr;
6072: else break;
6073: }
6074:
6075: /* Check validity of \R options. */
6076:
6077: switch (options & (PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE))
6078: {
6079: case 0:
6080: case PCRE_BSR_ANYCRLF:
6081: case PCRE_BSR_UNICODE:
6082: break;
6083: default: errorcode = ERR56; goto PCRE_EARLY_ERROR_RETURN;
6084: }
6085:
6086: /* Handle different types of newline. The three bits give seven cases. The
6087: current code allows for fixed one- or two-byte sequences, plus "any" and
6088: "anycrlf". */
6089:
6090: switch (options & PCRE_NEWLINE_BITS)
6091: {
6092: case 0: newline = NEWLINE; break; /* Build-time default */
6093: case PCRE_NEWLINE_CR: newline = '\r'; break;
6094: case PCRE_NEWLINE_LF: newline = '\n'; break;

```

```

6095: case PCRE_NEWLINE_CR+
6096:     PCRE_NEWLINE_LF: newline = (' \r' << 8) | ' \n'; break;
6097: case PCRE_NEWLINE_ANY: newline = -1; break;
6098: case PCRE_NEWLINE_ANYCRLF: newline = -2; break;
6099: default: errorcode = ERR56; goto PCRE_EARLY_ERROR_RETURN;
6100: }
6101:
6102: if (newline == -2)
6103: {
6104:     cd->nlttype = NLTYPE_ANYCRLF;
6105: }
6106: else if (newline < 0)
6107: {
6108:     cd->nlttype = NLTYPE_ANY;
6109: }
6110: else
6111: {
6112:     cd->nlttype = NLTYPE_FIXED;
6113:     if (newline > 255)
6114:     {
6115:         cd->nllen = 2;
6116:         cd->nl[0] = (newline >> 8) & 255;
6117:         cd->nl[1] = newline & 255;
6118:     }
6119:     else
6120:     {
6121:         cd->nllen = 1;
6122:         cd->nl[0] = newline;
6123:     }
6124: }
6125:
6126: /* Maximum back reference and backref bitmap. The bitmap records up to 31 back
6127: references to help in deciding whether (.*?) can be treated as anchored or not.
6128: */
6129:
6130: cd->top_backref = 0;
6131: cd->backref_map = 0;
6132:
6133: /* Reflect pattern for debugging output */
6134:
6135: DPRINTF(("----- \n"));

```



```

6136: DPRINTF((" %s \n", pattern));
6137:
6138: /* Pretend to compile the pattern while actually just accumulating the length
6139: of memory required. This behaviour is triggered by passing a non-NULL final
6140: argument to compile_regex(). We pass a block of workspace (cworkspace) for it
6141: to compile parts of the pattern into; the compiled code is discarded when it is
6142: no longer needed, so hopefully this workspace will never overflow, though there
6143: is a test for its doing so. */
6144:
6145: cd->bracount = cd->final_bracount = 0;
6146: cd->names_found = 0;
6147: cd->name_entry_size = 0;
6148: cd->name_table = NULL;
6149: cd->start_workspace = cworkspace;
6150: cd->start_code = cworkspace;
6151: cd->hwm = cworkspace;
6152: cd->start_pattern = (const uschar *)pattern;
6153: cd->end_pattern = (const uschar *) (pattern + strlen(pattern));
6154: cd->req_varyopt = 0;
6155: cd->external_options = options;
6156: cd->external_flags = 0;
6157:
6158: /* Now do the pre-compile. On error, errorcode will be set non-zero, so we
6159: don't need to look at the result of the function here. The initial options have
6160: been put into the cd block so that they can be changed if an option setting is
6161: found within the regex right at the beginning. Bringing initial option settings
6162: outside can help speed up starting point checks. */
6163:
6164: ptr += skipatstart;
6165: code = cworkspace;
6166: *code = OP_BRA;
6167: (void)compile_regex(cd->external_options, cd->external_options & PCRE_IMS,
6168: &code, &ptr, &errorcode, FALSE, FALSE, 0, &firstbyte, &reqbyte, NULL, cd,
6169: &length);
6170: if (errorcode != 0) goto PCRE_EARLY_ERROR_RETURN;
6171:
6172: DPRINTF(("end pre-compile: length=%d workspace=%d \n", length,
6173: cd->hwm - cworkspace));
6174:
6175: if (length > MAX_PATTERN_SIZE)
6176: {

```

```

6177: errorcode = ERR20;
6178: goto PCRE_EARLY_ERROR_RETURN;
6179: }
6180:
6181: /* Compute the size of data block needed and get it, either from malloc or
6182: externally provided function. Integer overflow should no longer be possible
6183: because nowadays we limit the maximum value of cd->names_found and
6184: cd->name_entry_size. */
6185:
6186: size = length + sizeof(real_pcre) + cd->names_found * (cd->name_entry_size + 3);
6187: re = (real_pcre *) (pcre_malloc)(size);
6188:
6189: if (re == NULL)
6190: {
6191:   errorcode = ERR21;
6192:   goto PCRE_EARLY_ERROR_RETURN;
6193: }
6194:
6195: /* Put in the magic number, and save the sizes, initial options, internal
6196: flags, and character table pointer. NULL is used for the default character
6197: tables. The nullpad field is at the end; it's there to help in the case when a
6198: regex compiled on a system with 4-byte pointers is run on another with 8-byte
6199: pointers. */
6200:
6201: re->magic_number = MAGIC_NUMBER;
6202: re->size = size;
6203: re->options = cd->external_options;
6204: re->flags = cd->external_flags;
6205: re->dummy1 = 0;
6206: re->first_byte = 0;
6207: re->req_byte = 0;
6208: re->name_table_offset = sizeof(real_pcre);
6209: re->name_entry_size = cd->name_entry_size;
6210: re->name_count = cd->names_found;
6211: re->ref_count = 0;
6212: re->tables = (tables == _pcre_default_tables)? NULL : tables;
6213: re->nullpad = NULL;
6214:
6215: /* The starting points of the name/number translation table and of the code are
6216: passed around in the compile data block. The start/end pattern and initial
6217: options are already set from the pre-compile phase, as is the name_entry_size

```

```

6218: field. Reset the bracket count and the names_found field. Also reset the hwm
6219: field; this time it's used for remembering forward references to subpatterns.
6220: */
6221:
6222: cd->final_bracount = cd->bracount; /* Save for checking forward references */
6223: cd->bracount = 0;
6224: cd->names_found = 0;
6225: cd->name_table = (uschar *)re + re->name_table_offset;
6226: codestart = cd->name_table + re->name_entry_size * re->name_count;
6227: cd->start_code = codestart;
6228: cd->hwm = cworkspace;
6229: cd->req_varyopt = 0;
6230: cd->had_accept = FALSE;
6231:
6232: /* Set up a starting, non-extracting bracket, then compile the expression. On
6233: error, errorcode will be set non-zero, so we don't need to look at the result
6234: of the function here. */
6235:
6236: ptr = (const uschar *)pattern + skipatstart;
6237: code = (uschar *)codestart;
6238: *code = OP_BRA;
6239: (void)compile_regex(re->options, re->options & PCRE_IMS, &code, &ptr,
6240: &errorcode, FALSE, FALSE, 0, &firstbyte, &reqbyte, NULL, cd, NULL);
6241: re->top_bracket = cd->bracount;
6242: re->top_backref = cd->top_backref;
6243: re->flags = cd->external_flags;
6244:
6245: if (cd->had_accept) reqbyte = -1; /* Must disable after (*ACCEPT) */
6246:
6247: /* If not reached end of pattern on success, there's an excess bracket. */
6248:
6249: if (errorcode == 0 && *ptr != 0) errorcode = ERR22;
6250:
6251: /* Fill in the terminating state and check for disastrous overflow, but
6252: if debugging, leave the test till after things are printed out. */
6253:
6254: *code++ = OP_END;
6255:
6256: #ifndef DEBUG
6257: if (code - codestart > length) errorcode = ERR23;
6258: #endif

```

```

6259:
6260: /* Fill in any forward references that are required. */
6261:
6262: while (errorcode == 0 && cd->hwm > cworkspace)
6263: {
6264:     int offset, recno;
6265:     const uschar *groupptr;
6266:     cd->hwm -= LINK_SIZE;
6267:     offset = GET(cd->hwm, 0);
6268:     recno = GET(codestart, offset);
6269:     groupptr = find_bracket(codestart, (re->options & PCRE_UTF8) != 0, recno);
6270:     if (groupptr == NULL) errorcode = ERR53;
6271:     else PUT(((uschar *)codestart), offset, groupptr - codestart);
6272: }
6273:
6274: /* Give an error if there's back reference to a non-existent capturing
6275: subpattern. */
6276:
6277: if (errorcode == 0 && re->top_backref > re->top_bracket) errorcode = ERR15;
6278:
6279: /* Failed to compile, or error while post-processing */
6280:
6281: if (errorcode != 0)
6282: {
6283:     (pcre_free)(re);
6284:     PCRE_EARLY_ERROR_RETURN:
6285:     *erroroffset = ptr - (const uschar *)pattern;
6286:     PCRE_EARLY_ERROR_RETURN2:
6287:     *errorptr = find_error_text(errorcode);
6288:     if (errorcodeptr != NULL) *errorcodeptr = errorcode;
6289:     return NULL;
6290: }
6291:
6292: /* If the anchored option was not passed, set the flag if we can determine that
6293: the pattern is anchored by virtue of ^ characters or \A or anything else (such
6294: as starting with .* when DOTALL is set).
6295:
6296: Otherwise, if we know what the first byte has to be, save it, because that
6297: speeds up unanchored matches no end. If not, see if we can set the
6298: PCRE_STARTLINE flag. This is helpful for multiline matches when all branches
6299: start with ^, and also when all branches start with .* for non-DOTALL matches.

```

```

6300: */
6301:
6302: if ((re->options & PCRE_ANCHORED) == 0)
6303: {
6304:   int temp_options = re->options; /* May get changed during these scans */
6305:   if (is_anchored(codestart, &temp_options, 0, cd->backref_map))
6306:     re->options |= PCRE_ANCHORED;
6307:   else
6308:   {
6309:     if (firstbyte < 0)
6310:       firstbyte = find_firstassertedchar(codestart, &temp_options, FALSE);
6311:     if (firstbyte >= 0) /* Remove caseless flag for non-caseable chars */
6312:     {
6313:       int ch = firstbyte & 255;
6314:       re->first_byte = ((firstbyte & REQ_CASELESS) != 0 &&
6315:         cd->fcc[ch] == ch)? ch : firstbyte;
6316:       re->flags |= PCRE_FIRSTSET;
6317:     }
6318:     else if (is_startline(codestart, 0, cd->backref_map))
6319:       re->flags |= PCRE_STARTLINE;
6320:   }
6321: }
6322:
6323: /* For an anchored pattern, we use the "required byte" only if it follows a
6324: variable length item in the regex. Remove the caseless flag for non-caseable
6325: bytes. */
6326:
6327: if (reqbyte >= 0 &&
6328:   ((re->options & PCRE_ANCHORED) == 0 || (reqbyte & REQ_VARY) != 0))
6329: {
6330:   int ch = reqbyte & 255;
6331:   re->req_byte = ((reqbyte & REQ_CASELESS) != 0 &&
6332:     cd->fcc[ch] == ch)? (reqbyte & ~REQ_CASELESS) : reqbyte;
6333:   re->flags |= PCRE_REQCHSET;
6334: }
6335:
6336: /* Print out the compiled data if debugging is enabled. This is never the
6337: case when building a production library. */
6338:
6339: #ifdef DEBUG
6340:

```

```

6341: printf("Length = %d top_bracket = %d top_backref = %d \n",
6342:  length, re->top_bracket, re->top_backref);
6343:
6344: printf("Options=%08x \n", re->options);
6345:
6346: if ((re->flags & PCRE_FIRSTSET) != 0)
6347: {
6348:  int ch = re->first_byte & 255;
6349:  const char *caseless = ((re->first_byte & REQ_CASELESS) == 0)?
6350:   "" : " (caseless)";
6351:  if (isprint(ch)) printf("First char = %c%s \n", ch, caseless);
6352:  else printf("First char = \\x%02x%s \n", ch, caseless);
6353: }
6354:
6355: if ((re->flags & PCRE_REQCHSET) != 0)
6356: {
6357:  int ch = re->req_byte & 255;
6358:  const char *caseless = ((re->req_byte & REQ_CASELESS) == 0)?
6359:   "" : " (caseless)";
6360:  if (isprint(ch)) printf("Req char = %c%s \n", ch, caseless);
6361:  else printf("Req char = \\x%02x%s \n", ch, caseless);
6362: }
6363:
6364: pcre_printint(re, stdout, TRUE);
6365:
6366: /* This check is done here in the debugging case so that the code that
6367: was compiled can be seen. */
6368:
6369: if (code - codestart > length)
6370: {
6371:  (pcre_free)(re);
6372:  *errorptr = find_error_text(ERR23);
6373:  *erroroffset = ptr - (uschar *)pattern;
6374:  if (errorcodeptr != NULL) *errorcodeptr = ERR23;
6375:  return NULL;
6376: }
6377: #endif /* DEBUG */
6378:
6379: return (pcre *)re;
6380: }
6381:

```

6382: /* End of pcre_compile.c */

File: sdm/VxWorks/libRegex/pcre_version.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:         Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```



```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_version(), which returns a
42: string that identifies the PCRE version that is in use. */
43:
44:
45: #ifdef HAVE_CONFIG_H
46: #include "config.h"
47: #endif
48:
49: #include "pcre_internal.h"
50:
51:
52: /*****
53:  *      Return version string      *
54:  *****/
55:
56: /* These macros are the standard way of turning unquoted text into C strings.
57: They allow macros like PCRE_MAJOR to be defined without quotes, which is
58: convenient for user programs that want to test its value. */
59:
60: #define STRING(a) #a
61: #define XSTRING(s) STRING(s)
62:
63: /* A problem turned up with PCRE_PRERELEASE, which is defined empty for
64: production releases. Originally, it was used naively in this code:
65:
66: return XSTRING(PCRE_MAJOR)
67:      "." XSTRING(PCRE_MINOR)
68:      XSTRING(PCRE_PRERELEASE)
69:      " " XSTRING(PCRE_DATE);
70:
71: However, when PCRE_PRERELEASE is empty, this leads to an attempted expansion of
72: STRING(). The C standard states: "If (before argument substitution) any
73: argument consists of no preprocessing tokens, the behavior is undefined." It
74: turns out the gcc treats this case as a single empty string - which is what we
75: really want - but Visual C grumbles about the lack of an argument for the
76: macro. Unfortunately, both are within their rights. To cope with both ways of

```

```

77: handling this, I had resort to some messy hackery that does a test at run time.
78: I could find no way of detecting that a macro is defined as an empty string at
79: pre-processor time. This hack uses a standard trick for avoiding calling
80: the STRING macro with an empty argument when doing the test. */
81:
82: PCRE_EXP_DEFN const char * PCRE_CALL_CONVENTION
83: pcre_version(void)
84: {
85: return (XSTRING(Z PCRE_PRERELEASE)[1] == 0)?
86: XSTRING(PCRE_MAJOR.PCRE_MINOR PCRE_DATE) :
87: XSTRING(PCRE_MAJOR.PCRE_MINOR) XSTRING(PCRE_PRERELEASE PCRE_DATE);
88: }
89:
90: /* End of pcre_version.c */

```

File: sdm/VxWorks/libRegex/pcre_refcount.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_refcount(), which is an
42: auxiliary function that can be used to maintain a reference count in a compiled
43: pattern data block. This might be helpful in applications where the block is
44: shared by different users. */
45:
46:
47: #ifdef HAVE_CONFIG_H
48: #include "config.h"
49: #endif
50:
51: #include "pcre_internal.h"
52:
53:
54: /*****
55: *      Maintain reference count      *
56: *****/
57:
58: /* The reference count is a 16-bit field, initialized to zero. It is not
59: possible to transfer a non-zero count from one host to a different host that
60: has a different byte order - though I can't see why anyone in their right mind
61: would ever want to do that!
62:
63: Arguments:
64:   argument_re  points to compiled code
65:   adjust      value to add to the count
66:
67: Returns:      the (possibly updated) count value (a non-negative number), or
68:               a negative error number
69: */
70:
71: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
72: pcre_refcount(pcre *argument_re, int adjust)
73: {
74:   real_pcre *re = (real_pcre *)argument_re;
75:   if (re == NULL) return PCRE_ERROR_NULL;
76:   re->ref_count = (-adjust > re->ref_count)? 0 :

```

```
77:         (adjust + re->ref_count > 65535)? 65535 :
78:         re->ref_count + adjust;
79: return re->ref_count;
80: }
81:
82: /* End of pcre_refcount.c */
```

File: sdm/VxWorks/libRegex/pcre_printint.src

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains a PCRE private debugging function for printing out the
42: internal form of a compiled regular expression, along with some supporting
43: local functions. This source file is used in two places:
44:
45: (1) It is #included by pcre_compile.c when it is compiled in debugging mode
46: (DEBUG defined in pcre_internal.h). It is not included in production compiles.
47:
48: (2) It is always #included by pcretest.c, which can be asked to print out a
49: compiled regex for debugging purposes. */
50:
51:
52: /* Macro that decides whether a character should be output as a literal or in
53: hexadecimal. We don't use isprint() because that can vary from system to system
54: (even without the use of locales) and we want the output always to be the same,
55: for testing purposes. This macro is used in pcretest as well as in this file. */
56:
57: #define PRINTABLE(c) ((c) >= 32 && (c) < 127)
58:
59: /* The table of operator names. */
60:
61: static const char *OP_names[] = { OP_NAME_LIST };
62:
63:
64:
65: /*****
66: *      Print single- or multi-byte character      *
67: *****/
68:
69: static int
70: print_char(FILE *f, uschar *ptr, BOOL utf8)
71: {
72: int c = *ptr;
73:
74: #ifndef SUPPORT_UTF8
75: utf8 = utf8; /* Avoid compiler warning */
76: if (PRINTABLE(c)) fprintf(f, "%c", c); else fprintf(f, " \\x%02x", c);

```

```

77: return 0;
78:
79: #else
80: if (!utf8 || (c & 0xc0) != 0xc0)
81: {
82:   if (PRINTABLE(c)) fprintf(f, "%c", c); else fprintf(f, "\\x%02x", c);
83:   return 0;
84: }
85: else
86: {
87:   int i;
88:   int a = _pcre_utf8_table4[c & 0x3f]; /* Number of additional bytes */
89:   int s = 6*a;
90:   c = (c & _pcre_utf8_table3[a]) << s;
91:   for (i = 1; i <= a; i++)
92:   {
93:     /* This is a check for malformed UTF-8; it should only occur if the sanity
94:      check has been turned off. Rather than swallow random bytes, just stop if
95:      we hit a bad one. Print it with \X instead of \x as an indication. */
96:
97:     if ((ptr[i] & 0xc0) != 0x80)
98:     {
99:       fprintf(f, "\\X{%x}", c);
100:      return i - 1;
101:     }
102:
103:     /* The byte is OK */
104:
105:     s -= 6;
106:     c |= (ptr[i] & 0x3f) << s;
107:   }
108:   if (c < 128) fprintf(f, "\\x%02x", c); else fprintf(f, "\\x{%x}", c);
109:   return a;
110: }
111: #endif
112: }
113:
114:
115:
116: /*****
117: *      Find Unicode property name      *

```



```

118: *****/
119:
120: static const char *
121: get_ucpname(int ptype, int pvalue)
122: {
123: #ifdef SUPPORT_UCP
124: int i;
125: for (i = _pcre_utt_size - 1; i >= 0; i--)
126: {
127: if (ptype == _pcre_utt[i].type && pvalue == _pcre_utt[i].value) break;
128: }
129: return (i >= 0)? _pcre_utt_names + _pcre_utt[i].name_offset : "??";
130: #else
131: /* It gets harder and harder to shut off unwanted compiler warnings. */
132: ptype = ptype * pvalue;
133: return (ptype == pvalue)? "??": "??";
134: #endif
135: }
136:
137:
138:
139: *****/
140: *      Print compiled regex      *
141: *****/
142:
143: /* Make this function work for a regex with integers either byte order.
144: However, we assume that what we are passed is a compiled regex. The
145: print_lengths flag controls whether offsets and lengths of items are printed.
146: They can be turned off from pcretest so that automatic tests on bytecode can be
147: written that do not depend on the value of LINK_SIZE. */
148:
149: static void
150: pcre_printint(pcre *external_re, FILE *f, BOOL print_lengths)
151: {
152: real_pcre *re = (real_pcre *)external_re;
153: uschar *codestart, *code;
154: BOOL utf8;
155:
156: unsigned int options = re->options;
157: int offset = re->name_table_offset;
158: int count = re->name_count;

```

```

159: int size = re->name_entry_size;
160:
161: if (re->magic_number != MAGIC_NUMBER)
162: {
163: offset = ((offset << 8) & 0xff00) | ((offset >> 8) & 0xff);
164: count = ((count << 8) & 0xff00) | ((count >> 8) & 0xff);
165: size = ((size << 8) & 0xff00) | ((size >> 8) & 0xff);
166: options = ((options << 24) & 0xff000000) |
167:           ((options << 8) & 0x00ff0000) |
168:           ((options >> 8) & 0x0000ff00) |
169:           ((options >> 24) & 0x000000ff);
170: }
171:
172: code = codestart = (uchar *)re + offset + count * size;
173: utf8 = (options & PCRE_UTF8) != 0;
174:
175: for(;;)
176: {
177: uchar *ccode;
178: int c;
179: int extra = 0;
180:
181: if (print_lengths)
182:   fprintf(f, "%3d ", (int)(code - codestart));
183: else
184:   fprintf(f, " ");
185:
186: switch(*code)
187: {
188: case OP_END:
189:   fprintf(f, "   %s\n", OP_names[*code]);
190:   fprintf(f, "-----\n");
191:   return;
192:
193: case OP_OPT:
194:   fprintf(f, " %.2x %s", code[1], OP_names[*code]);
195:   break;
196:
197: case OP_CHAR:
198:   fprintf(f, " ");
199:   do

```

```

200:  {
201:    code++;
202:    code += 1 + print_char(f, code, utf8);
203:  }
204:  while (*code == OP_CHAR);
205:  fprintf(f, "\n");
206:  continue;
207:
208:  case OP_CHARNC:
209:    fprintf(f, " NC ");
210:    do
211:    {
212:      code++;
213:      code += 1 + print_char(f, code, utf8);
214:    }
215:    while (*code == OP_CHARNC);
216:    fprintf(f, "\n");
217:    continue;
218:
219:  case OP_CBRA:
220:  case OP_SCBRA:
221:    if (print_lengths) fprintf(f, "%3d ", GET(code, 1));
222:    else fprintf(f, " ");
223:    fprintf(f, "%s %d", OP_names[*code], GET2(code, 1+LINK_SIZE));
224:    break;
225:
226:  case OP_BRA:
227:  case OP_SBRA:
228:  case OP_KETRMX:
229:  case OP_KETRMN:
230:  case OP_ALT:
231:  case OP_KET:
232:  case OP_ASSERT:
233:  case OP_ASSERT_NOT:
234:  case OP_ASSERTBACK:
235:  case OP_ASSERTBACK_NOT:
236:  case OP_ONCE:
237:  case OP_COND:
238:  case OP_SCOND:
239:  case OP_REVERSE:
240:    if (print_lengths) fprintf(f, "%3d ", GET(code, 1));

```

```

241:     else fprintf(f, " ");
242:     fprintf(f, "%s", OP_names[*code]);
243:     break;
244:
245:     case OP_CREF:
246:         fprintf(f, "%3d %s", GET2(code,1), OP_names[*code]);
247:         break;
248:
249:     case OP_RREF:
250:         c = GET2(code, 1);
251:         if (c == RREF_ANY)
252:             fprintf(f, "    Cond recurse any");
253:         else
254:             fprintf(f, "    Cond recurse %d", c);
255:         break;
256:
257:     case OP_DEF:
258:         fprintf(f, "    Cond def");
259:         break;
260:
261:     case OP_STAR:
262:     case OP_MINSTAR:
263:     case OP_POSSTAR:
264:     case OP_PLUS:
265:     case OP_MINPLUS:
266:     case OP_POSPLUS:
267:     case OP_QUERY:
268:     case OP_MINQUERY:
269:     case OP_POSQUERY:
270:     case OP_TYPESTAR:
271:     case OP_TYPEMINSTAR:
272:     case OP_TYPEPOSSTAR:
273:     case OP_TYPEPLUS:
274:     case OP_TYPEMINPLUS:
275:     case OP_TYPEPOSPLUS:
276:     case OP_TYPEQUERY:
277:     case OP_TYPEMINQUERY:
278:     case OP_TYPEPOSQUERY:
279:         fprintf(f, " ");
280:         if (*code >= OP_TYPESTAR)
281:             {

```

```

282:     fprintf(f, "%s", OP_names[code[1]]);
283:     if (code[1] == OP_PROP || code[1] == OP_NOTPROP)
284:     {
285:         fprintf(f, " %s ", get_ucpname(code[2], code[3]));
286:         extra = 2;
287:     }
288: }
289: else extra = print_char(f, code+1, utf8);
290: fprintf(f, "%s", OP_names[*code]);
291: break;
292:
293: case OP_EXACT:
294: case OP_UPTO:
295: case OP_MINUPTO:
296: case OP_POSUPTO:
297:     fprintf(f, " ");
298:     extra = print_char(f, code+3, utf8);
299:     fprintf(f, "{}");
300:     if (*code != OP_EXACT) fprintf(f, "0,");
301:     fprintf(f, "%d}", GET2(code,1));
302:     if (*code == OP_MINUPTO) fprintf(f, "?");
303:     else if (*code == OP_POSUPTO) fprintf(f, "+");
304:     break;
305:
306: case OP_TYPEEXACT:
307: case OP_TYPEUPTO:
308: case OP_TYPEMINUPTO:
309: case OP_TYPEPOSUPTO:
310:     fprintf(f, " %s", OP_names[code[3]]);
311:     if (code[3] == OP_PROP || code[3] == OP_NOTPROP)
312:     {
313:         fprintf(f, " %s ", get_ucpname(code[4], code[5]));
314:         extra = 2;
315:     }
316:     fprintf(f, "{}");
317:     if (*code != OP_TYPEEXACT) fprintf(f, "0,");
318:     fprintf(f, "%d}", GET2(code,1));
319:     if (*code == OP_TYPEMINUPTO) fprintf(f, "?");
320:     else if (*code == OP_TYPEPOSUPTO) fprintf(f, "+");
321:     break;
322:

```

```

323: case OP_NOT:
324: c = code[1];
325: if (PRINTABLE(c)) fprintf(f, "  [^%c]", c);
326:   else fprintf(f, "  [^\\x%02x]", c);
327: break;
328:
329: case OP_NOTSTAR:
330: case OP_NOTMINSTAR:
331: case OP_NOTPOSSTAR:
332: case OP_NOTPLUS:
333: case OP_NOTMINPLUS:
334: case OP_NOTPOSPLUS:
335: case OP_NOTQUERY:
336: case OP_NOTMINQUERY:
337: case OP_NOTPOSQUERY:
338: c = code[1];
339: if (PRINTABLE(c)) fprintf(f, "  [^%c]", c);
340:   else fprintf(f, "  [^\\x%02x]", c);
341: fprintf(f, "%s", OP_names[*code]);
342: break;
343:
344: case OP_NOTEXACT:
345: case OP_NOTUPTO:
346: case OP_NOTMINUPTO:
347: case OP_NOTPOSUPTO:
348: c = code[3];
349: if (PRINTABLE(c)) fprintf(f, "  [^%c]{", c);
350:   else fprintf(f, "  [^\\x%02x]{", c);
351: if (*code != OP_NOTEXACT) fprintf(f, "0,");
352: fprintf(f, "%d}", GET2(code,1));
353: if (*code == OP_NOTMINUPTO) fprintf(f, "?");
354:   else if (*code == OP_NOTPOSUPTO) fprintf(f, "+");
355: break;
356:
357: case OP_RECURSE:
358: if (print_lengths) fprintf(f, "%3d ", GET(code, 1));
359:   else fprintf(f, "  ");
360: fprintf(f, "%s", OP_names[*code]);
361: break;
362:
363: case OP_REF:

```

```

364: fprintf(f, "  \\%d", GET2(code,1));
365: ccode = code + _pcre_OP_lengths[*code];
366: goto CLASS_REF_REPEAT;
367:
368: case OP_CALLOUT:
369: fprintf(f, "  %s %d %d %d", OP_names[*code], code[1], GET(code,2),
370:   GET(code, 2 + LINK_SIZE));
371: break;
372:
373: case OP_PROP:
374: case OP_NOTPROP:
375: fprintf(f, "  %s %s", OP_names[*code], get_ucpname(code[1], code[2]));
376: break;
377:
378: /* OP_XCLASS can only occur in UTF-8 mode. However, there's no harm in
379: having this code always here, and it makes it less messy without all those
380: #ifdefs. */
381:
382: case OP_CLASS:
383: case OP_NCLASS:
384: case OP_XCLASS:
385: {
386:   int i, min, max;
387:   BOOL printmap;
388:
389:   fprintf(f, "  [");
390:
391:   if (*code == OP_XCLASS)
392:   {
393:     extra = GET(code, 1);
394:     ccode = code + LINK_SIZE + 1;
395:     printmap = (*ccode & XCL_MAP) != 0;
396:     if ((*ccode++ & XCL_NOT) != 0) fprintf(f, "^");
397:   }
398:   else
399:   {
400:     printmap = TRUE;
401:     ccode = code + 1;
402:   }
403:
404:   /* Print a bit map */

```

```

405:
406:   if (printmap)
407:   {
408:       for (i = 0; i < 256; i++)
409:       {
410:           if ((ccode[i/8] & (1 << (i&7))) != 0)
411:           {
412:               int j;
413:               for (j = i+1; j < 256; j++)
414:                   if ((ccode[j/8] & (1 << (j&7))) == 0) break;
415:               if (i == '-' || i == ']') fprintf(f, "\\");
416:               if (PRINTABLE(i)) fprintf(f, "%c", i);
417:               else fprintf(f, "\\x%02x", i);
418:               if (--j > i)
419:               {
420:                   if (j != i + 1) fprintf(f, "-");
421:                   if (j == '-' || j == ']') fprintf(f, "\\");
422:                   if (PRINTABLE(j)) fprintf(f, "%c", j);
423:                   else fprintf(f, "\\x%02x", j);
424:               }
425:               i = j;
426:           }
427:       }
428:       ccode += 32;
429:   }
430:
431:   /* For an XCLASS there is always some additional data */
432:
433:   if (*code == OP_XCLASS)
434:   {
435:       int ch;
436:       while ((ch = *ccode++) != XCL_END)
437:       {
438:           if (ch == XCL_PROP)
439:           {
440:               int ptype = *ccode++;
441:               int pvalue = *ccode++;
442:               fprintf(f, "\\p{%s}", get_ucpname(ptype, pvalue));
443:           }
444:           else if (ch == XCL_NOTPROP)
445:           {

```



```

446:     int ptype = *ccode++;
447:     int pvalue = *ccode++;
448:     fprintf(f, "\\P{%s}", get_ucpname(ptype, pvalue));
449: }
450: else
451: {
452:     ccode += 1 + print_char(f, ccode, TRUE);
453:     if (ch == XCL_RANGE)
454:     {
455:         fprintf(f, "-");
456:         ccode += 1 + print_char(f, ccode, TRUE);
457:     }
458: }
459: }
460: }
461:
462: /* Indicate a non-UTF8 class which was created by negation */
463:
464: fprintf(f, "]%s", (*code == OP_NCLASS)? " (neg)" : "");
465:
466: /* Handle repeats after a class or a back reference */
467:
468: CLASS_REF_REPEAT:
469: switch(*ccode)
470: {
471:     case OP_CRSTAR:
472:     case OP_CRMINSTAR:
473:     case OP_CRPLUS:
474:     case OP_CRMINPLUS:
475:     case OP_CRQUERY:
476:     case OP_CRMINQUERY:
477:         fprintf(f, "%s", OP_names[*ccode]);
478:         extra += _pcre_OP_lengths[*ccode];
479:         break;
480:
481:     case OP_CRRANGE:
482:     case OP_CRMINRANGE:
483:         min = GET2(ccode,1);
484:         max = GET2(ccode,3);
485:         if (max == 0) fprintf(f, "{%d,}", min);
486:         else fprintf(f, "{%d,%d}", min, max);

```

```

487:     if (*ccode == OP_CRMINRANGE) fprintf(f, "?");
488:     extra += _pcre_OP_lengths[*ccode];
489:     break;
490:
491:     /* Do nothing if it's not a repeat; this code stops picky compilers
492:     warning about the lack of a default code path. */
493:
494:     default:
495:     break;
496:     }
497: }
498: break;
499:
500: /* Anything else is just an item with no data*/
501:
502: default:
503: fprintf(f, "  %s", OP_names[*code]);
504: break;
505: }
506:
507: code += _pcre_OP_lengths[*code] + extra;
508: fprintf(f, "\n");
509: }
510: }
511:
512: /* End of pcre_printint.src */

```

File: sdm/VxWorks/libRegex/pcre_study.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains the external function pcre_study(), along with local
42: supporting functions. */
43:
44:
45: #ifdef HAVE_CONFIG_H
46: #include "config.h"
47: #endif
48:
49: #include "pcre_internal.h"
50:
51:
52: /* Returns from set_start_bits() */
53:
54: enum { SSB_FAIL, SSB_DONE, SSB_CONTINUE };
55:
56:
57: /*******
58: *      Set a bit and maybe its alternate case      *
59: *****/
60:
61: /* Given a character, set its bit in the table, and also the bit for the other
62: version of a letter if we are caseless.
63:
64: Arguments:
65:  start_bits  points to the bit map
66:  c           is the character
67:  caseless    the caseless flag
68:  cd          the block with char table pointers
69:
70: Returns:     nothing
71: */
72:
73: static void
74: set_bit(uschar *start_bits, unsigned int c, BOOL caseless, compile_data *cd)
75: {
76: start_bits[c/8] |= (1 << (c&7));

```

```

77: if (caseless && (cd->ctypes[c] & ctype_letter) != 0)
78:   start_bits[cd->fcc[c]/8] |= (1 << (cd->fcc[c]&7));
79: }
80:
81:
82:
83: /*****
84:  *      Create bitmap of starting bytes      *
85:  *****/
86:
87: /* This function scans a compiled unanchored expression recursively and
88: attempts to build a bitmap of the set of possible starting bytes. As time goes
89: by, we may be able to get more clever at doing this. The SSB_CONTINUE return is
90: useful for parenthesized groups in patterns such as (a*)b where the group
91: provides some optional starting bytes but scanning must continue at the outer
92: level to find at least one mandatory byte. At the outermost level, this
93: function fails unless the result is SSB_DONE.
94:
95: Arguments:
96:   code      points to an expression
97:   start_bits points to a 32-byte table, initialized to 0
98:   caseless  the current state of the caseless flag
99:   utf8      TRUE if in UTF-8 mode
100:   cd        the block with char table pointers
101:
102: Returns:    SSB_FAIL    => Failed to find any starting bytes
103:             SSB_DONE    => Found mandatory starting bytes
104:             SSB_CONTINUE => Found optional starting bytes
105: */
106:
107: static int
108: set_start_bits(const uschar *code, uschar *start_bits, BOOL caseless,
109:   BOOL utf8, compile_data *cd)
110: {
111:   register int c;
112:   int yield = SSB_DONE;
113:
114:   #if 0
115:
116:   /* The following comment and code was inserted in January 1999. In May 2006,

```

```

117: when it was observed to cause compiler warnings about unused values, I took it
118: out again. If anybody is still using OS/2, they will have to put it back
119: manually. */
120:
121: /* This next statement and the later reference to dummy are here in order to
122: trick the optimizer of the IBM C compiler for OS/2 into generating correct
123: code. Apparently IBM isn't going to fix the problem, and we would rather not
124: disable optimization (in this module it actually makes a big difference, and
125: the pcre module can use all the optimization it can get). */
126:
127: volatile int dummy;
128:
===== /*
129: #endif
130:
131: do
132: {
133: const uschar *tcode = code + (((int)*code == OP_CBRA)? 3:1) + LINK_SIZE;
134: BOOL try_next = TRUE;
135:
136: while (try_next) /* Loop for items in this branch */
137: {
138: int rc;
139: switch(*tcode)
140: {
141: /* Fail if we reach something we don't understand */
142:
143: default:
144: return SSB_FAIL;
145:
146: /* If we hit a bracket or a positive lookahead assertion, recurse to set
147: bits from within the subpattern. If it can't find anything, we have to
148: give up. If it finds some mandatory character(s), we are done for this
149: branch. Otherwise, carry on scanning after the subpattern. */
150:
151: case OP_BRA:
152: case OP_SBRA:
153: case OP_CBRA:
154: case OP_SCBRA:
155: case OP_ONCE:
156: case OP_ASSERT:

```

```

157: rc = set_start_bits(tcode, start_bits, caseless, utf8, cd);
158: if (rc == SSB_FAIL) return SSB_FAIL;
159: if (rc == SSB_DONE) try_next = FALSE; else
160: {
161:   do tcode += GET(tcode, 1); while (*tcode == OP_ALT);
162:   tcode += 1 + LINK_SIZE;
163: }
164: break;
165:
166: /* If we hit ALT or KET, it means we haven't found anything mandatory in
167: this branch, though we might have found something optional. For ALT, we
168: continue with the next alternative, but we have to arrange that the final
169: result from subpattern is SSB_CONTINUE rather than SSB_DONE. For KET,
170: return SSB_CONTINUE: if this is the top level, that indicates failure,
171: but after a nested subpattern, it causes scanning to continue. */
172:
173: case OP_ALT:
174:   yield = SSB_CONTINUE;
175:   try_next = FALSE;
176:   break;
177:
178: case OP_KET:
179: case OP_KETRMX:
180: case OP_KETRMN:
181:   return SSB_CONTINUE;
182:
183: /* Skip over callout */
184:
185: case OP_CALLOUT:
186:   tcode += 2 + 2*LINK_SIZE;
187:   break;
188:
189: /* Skip over lookbehind and negative lookahead assertions */
190:
191: case OP_ASSERT_NOT:
192: case OP_ASSERTBACK:
193: case OP_ASSERTBACK_NOT:
194:   do tcode += GET(tcode, 1); while (*tcode == OP_ALT);
195:   tcode += 1 + LINK_SIZE;
196:   break;
197:

```

```

198:  /* Skip over an option setting, changing the caseless flag */
199:
200:  case OP_OPT:
201:  caseless = (tcode[1] & PCRE_CASELESS) != 0;
202:  tcode += 2;
203:  break;
204:
205:  /* BRAZERO does the bracket, but carries on. */
206:
207:  case OP_BRAZERO:
208:  case OP_BRAMINZERO:
209:  if (set_start_bits(++tcode, start_bits, caseless, utf8, cd) == SSB_FAIL)
210:    return SSB_FAIL;
211:
=====
212:  See the comment at the head of this function concerning the next line,
213:  which was an old fudge for the benefit of OS/2.
214:  dummy = 1;
215:
===== */
216:  do tcode += GET(tcode,1); while (*tcode == OP_ALT);
217:  tcode += 1 + LINK_SIZE;
218:  break;
219:
220:  /* SKIPZERO skips the bracket. */
221:
222:  case OP_SKIPZERO:
223:  tcode++;
224:  do tcode += GET(tcode,1); while (*tcode == OP_ALT);
225:  tcode += 1 + LINK_SIZE;
226:  break;
227:
228:  /* Single-char * or ? sets the bit and tries the next item */
229:
230:  case OP_STAR:
231:  case OP_MINSTAR:
232:  case OP_POSSTAR:
233:  case OP_QUERY:
234:  case OP_MINQUERY:
235:  case OP_POSQUERY:
236:  set_bit(start_bits, tcode[1], caseless, cd);

```



```

237:     tcode += 2;
238: #ifdef SUPPORT_UTF8
239:     if (utf8 && tcode[-1] >= 0xc0)
240:         tcode += _pcre_utf8_table4[tcode[-1] & 0x3f];
241: #endif
242:     break;
243:
244:     /* Single-char upto sets the bit and tries the next */
245:
246:     case OP_UPTO:
247:     case OP_MINUPTO:
248:     case OP_POSUPTO:
249:         set_bit(start_bits, tcode[3], caseless, cd);
250:         tcode += 4;
251: #ifdef SUPPORT_UTF8
252:         if (utf8 && tcode[-1] >= 0xc0)
253:             tcode += _pcre_utf8_table4[tcode[-1] & 0x3f];
254: #endif
255:         break;
256:
257:     /* At least one single char sets the bit and stops */
258:
259:     case OP_EXACT:      /* Fall through */
260:         tcode += 2;
261:
262:     case OP_CHAR:
263:     case OP_CHARNC:
264:     case OP_PLUS:
265:     case OP_MINPLUS:
266:     case OP_POSPLUS:
267:         set_bit(start_bits, tcode[1], caseless, cd);
268:         try_next = FALSE;
269:         break;
270:
271:     /* Single character type sets the bits and stops */
272:
273:     case OP_NOT_DIGIT:
274:         for (c = 0; c < 32; c++)
275:             start_bits[c] |= ~cd->cbits[c+cbit_digit];
276:         try_next = FALSE;
277:         break;

```

```

278:
279:     case OP_DIGIT:
280:         for (c = 0; c < 32; c++)
281:             start_bits[c] |= cd->cbits[c+cbit_digit];
282:         try_next = FALSE;
283:         break;
284:
285:         /* The cbit_space table has vertical tab as whitespace; we have to
286:         discard it. */
287:
288:     case OP_NOT_WHITESPACE:
289:         for (c = 0; c < 32; c++)
290:             {
291:                 int d = cd->cbits[c+cbit_space];
292:                 if (c == 1) d &= ~0x08;
293:                 start_bits[c] |= ~d;
294:             }
295:         try_next = FALSE;
296:         break;
297:
298:         /* The cbit_space table has vertical tab as whitespace; we have to
299:         discard it. */
300:
301:     case OP_WHITESPACE:
302:         for (c = 0; c < 32; c++)
303:             {
304:                 int d = cd->cbits[c+cbit_space];
305:                 if (c == 1) d &= ~0x08;
306:                 start_bits[c] |= d;
307:             }
308:         try_next = FALSE;
309:         break;
310:
311:     case OP_NOT_WORDCHAR:
312:         for (c = 0; c < 32; c++)
313:             start_bits[c] |= ~cd->cbits[c+cbit_word];
314:         try_next = FALSE;
315:         break;
316:
317:     case OP_WORDCHAR:
318:         for (c = 0; c < 32; c++)

```

```

319:     start_bits[c] |= cd->cbits[c+cbit_word];
320:     try_next = FALSE;
321:     break;
322:
323:     /* One or more character type fudges the pointer and restarts, knowing
324:     it will hit a single character type and stop there. */
325:
326:     case OP_TYPEPLUS:
327:     case OP_TYPEMINPLUS:
328:     tcode++;
329:     break;
330:
331:     case OP_TYPEEXACT:
332:     tcode += 3;
333:     break;
334:
335:     /* Zero or more repeats of character types set the bits and then
336:     try again. */
337:
338:     case OP_TYPEUPTO:
339:     case OP_TYPEMINUPTO:
340:     case OP_TYPEPOSUPTO:
341:     tcode += 2;          /* Fall through */
342:
343:     case OP_TYPESTAR:
344:     case OP_TYPEMINSTAR:
345:     case OP_TYPEPOSSTAR:
346:     case OP_TYPEQUERY:
347:     case OP_TYPEMINQUERY:
348:     case OP_TYPEPOSQUERY:
349:     switch(tcode[1])
350:     {
351:     case OP_ANY:
352:     case OP_ALLANY:
353:     return SSB_FAIL;
354:
355:     case OP_NOT_DIGIT:
356:     for (c = 0; c < 32; c++)
357:     start_bits[c] |= ~cd->cbits[c+cbit_digit];
358:     break;
359:

```

```

360:     case OP_DIGIT:
361:         for (c = 0; c < 32; c++)
362:             start_bits[c] |= cd->cbits[c+cbit_digit];
363:         break;
364:
365:         /* The cbit_space table has vertical tab as whitespace; we have to
366:         discard it. */
367:
368:     case OP_NOT_WHITESPACE:
369:         for (c = 0; c < 32; c++)
370:             {
371:                 int d = cd->cbits[c+cbit_space];
372:                 if (c == 1) d &= ~0x08;
373:                 start_bits[c] |= ~d;
374:             }
375:         break;
376:
377:         /* The cbit_space table has vertical tab as whitespace; we have to
378:         discard it. */
379:
380:     case OP_WHITESPACE:
381:         for (c = 0; c < 32; c++)
382:             {
383:                 int d = cd->cbits[c+cbit_space];
384:                 if (c == 1) d &= ~0x08;
385:                 start_bits[c] |= d;
386:             }
387:         break;
388:
389:     case OP_NOT_WORDCHAR:
390:         for (c = 0; c < 32; c++)
391:             start_bits[c] |= ~cd->cbits[c+cbit_word];
392:         break;
393:
394:     case OP_WORDCHAR:
395:         for (c = 0; c < 32; c++)
396:             start_bits[c] |= cd->cbits[c+cbit_word];
397:         break;
398:     }
399:
400:     tcode += 2;

```

```

401:     break;
402:
403:     /* Character class where all the information is in a bit map: set the
404:     bits and either carry on or not, according to the repeat count. If it was
405:     a negative class, and we are operating with UTF-8 characters, any byte
406:     with a value >= 0xc4 is a potentially valid starter because it starts a
407:     character with a value > 255. */
408:
409:     case OP_NCLASS:
410: #ifdef SUPPORT_UTF8
411:         if (utf8)
412:         {
413:             start_bits[24] |= 0xf0;          /* Bits for 0xc4 - 0xc8 */
414:             memset(start_bits+25, 0xff, 7);  /* Bits for 0xc9 - 0xff */
415:         }
416: #endif
417:         /* Fall through */
418:
419:     case OP_CLASS:
420:         {
421:             tcode++;
422:
423:             /* In UTF-8 mode, the bits in a bit map correspond to character
424:             values, not to byte values. However, the bit map we are constructing is
425:             for byte values. So we have to do a conversion for characters whose
426:             value is > 127. In fact, there are only two possible starting bytes for
427:             characters in the range 128 - 255. */
428:
429: #ifdef SUPPORT_UTF8
430:             if (utf8)
431:             {
432:                 for (c = 0; c < 16; c++) start_bits[c] |= tcode[c];
433:                 for (c = 128; c < 256; c++)
434:                 {
435:                     if ((tcode[c/8] && (1 << (c&7))) != 0)
436:                     {
437:                         int d = (c >> 6) | 0xc0;      /* Set bit for this starter */
438:                         start_bits[d/8] |= (1 << (d&7)); /* and then skip on to the */
439:                         c = (c & 0xc0) + 0x40 - 1;      /* next relevant character. */
440:                     }
441:                 }

```

```

442:     }
443:
444:     /* In non-UTF-8 mode, the two bit maps are completely compatible. */
445:
446:     else
447: #endif
448:     {
449:         for (c = 0; c < 32; c++) start_bits[c] |= tcode[c];
450:     }
451:
452:     /* Advance past the bit map, and act on what follows */
453:
454:     tcode += 32;
455:     switch (*tcode)
456:     {
457:         case OP_CRSTAR:
458:         case OP_CRMINSTAR:
459:         case OP_CRQUERY:
460:         case OP_CRMINQUERY:
461:             tcode++;
462:             break;
463:
464:         case OP_CRRANGE:
465:         case OP_CRMINRANGE:
466:             if (((tcode[1] << 8) + tcode[2]) == 0) tcode += 5;
467:             else try_next = FALSE;
468:             break;
469:
470:         default:
471:             try_next = FALSE;
472:             break;
473:     }
474: }
475: break; /* End of bitmap class handling */
476:
477: } /* End of switch */
478: } /* End of try_next loop */
479:
480: code += GET(code, 1); /* Advance to next branch */
481: }
482: while (*code == OP_ALT);

```

```

483: return yield;
484: }
485:
486:
487:
488: /*****
489: *      Study a compiled expression      *
490: *****/
491:
492: /* This function is handed a compiled expression that it must study to produce
493: information that will speed up the matching. It returns a pcre_extra block
494: which then gets handed back to pcre_exec().
495:
496: Arguments:
497: re      points to the compiled expression
498: options contains option bits
499: errorptr points to where to place error messages;
500:         set NULL unless error
501:
502: Returns: pointer to a pcre_extra block, with study_data filled in and the
503:         appropriate flag set;
504:         NULL on error or if no optimization possible
505: */
506:
507: PCRE_EXP_DEFN pcre_extra * PCRE_CALL_CONVENTION
508: pcre_study(const pcre *external_re, int options, const char **errorptr)
509: {
510: uschar start_bits[32];
511: pcre_extra *extra;
512: pcre_study_data *study;
513: const uschar *tables;
514: uschar *code;
515: compile_data compile_block;
516: const real_pcre *re = (const real_pcre *)external_re;
517:
518: *errorptr = NULL;
519:
520: if (re == NULL || re->magic_number != MAGIC_NUMBER)
521: {
522: *errorptr = "argument is not a compiled regular expression";
523: return NULL;

```

```

524: }
525:
526: if ((options & ~PUBLIC_STUDY_OPTIONS) != 0)
527: {
528: *errorptr = "unknown or incorrect option bit(s) set";
529: return NULL;
530: }
531:
532: code = (uschar *)re + re->name_table_offset +
533: (re->name_count * re->name_entry_size);
534:
535: /* For an anchored pattern, or an unanchored pattern that has a first char, or
536: a multiline pattern that matches only at "line starts", no further processing
537: at present. */
538:
539: if ((re->options & PCRE_ANCHORED) != 0 ||
540: (re->flags & (PCRE_FIRSTSET|PCRE_STARTLINE)) != 0)
541: return NULL;
542:
543: /* Set the character tables in the block that is passed around */
544:
545: tables = re->tables;
546: if (tables == NULL)
547: (void)pcre_fullinfo(external_re, NULL, PCRE_INFO_DEFAULT_TABLES,
548: (void *)&tables);
549:
550: compile_block.lcc = tables + lcc_offset;
551: compile_block.fcc = tables + fcc_offset;
552: compile_block.cbits = tables + cbits_offset;
553: compile_block.ctypes = tables + ctypes_offset;
554:
555: /* See if we can find a fixed set of initial characters for the pattern. */
556:
557: memset(start_bits, 0, 32 * sizeof(uschar));
558: if (set_start_bits(code, start_bits, (re->options & PCRE_CASELESS) != 0,
559: (re->options & PCRE_UTF8) != 0, &compile_block) != SSB_DONE) return NULL;
560:
561: /* Get a pcre_extra block and a pcre_study_data block. The study data is put in
562: the latter, which is pointed to by the former, which may also get additional
563: data set later by the calling program. At the moment, the size of
564: pcre_study_data is fixed. We nevertheless save it in a field for returning via

```



```

565: the pcre_fullinfo() function so that if it becomes variable in the future, we
566: don't have to change that code. */
567:
568: extra = (pcre_extra *) (pcre_malloc
569: (sizeof(pcre_extra) + sizeof(pcre_study_data)));
570:
571: if (extra == NULL)
572: {
573: *errorptr = "failed to get memory";
574: return NULL;
575: }
576:
577: study = (pcre_study_data *) ((char *) extra + sizeof(pcre_extra));
578: extra->flags = PCRE_EXTRA_STUDY_DATA;
579: extra->study_data = study;
580:
581: study->size = sizeof(pcre_study_data);
582: study->options = PCRE_STUDY_MAPPED;
583: memcpy(study->start_bits, start_bits, sizeof(start_bits));
584:
585: return extra;
586: }
587:
588: /* End of pcre_study.c */

```

File: sdm/VxWorks/libRegex/pcre_tables.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
38: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
39: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains some fixed tables that are used by more than one of the
42: PCRE code modules. The tables are also #included by the pcretest program, which
43: uses macros to change their names from _pcre_xxx to xxxx, thereby avoiding name
44: clashes with the library. */
45:
46:
47: #ifdef HAVE_CONFIG_H
48: #include "config.h"
49: #endif
50:
51: #include "pcre_internal.h"
52:
53:
54: /* Table of sizes for the fixed-length opcodes. It's defined in a macro so that
55: the definition is next to the definition of the opcodes in pcre_internal.h. */
56:
57: const uschar _pcre_OP_lengths[] = { OP_LENGTHS };
58:
59:
60:
61: /*****
62: *      Tables for UTF-8 support      *
63: *****/
64:
65: /* These are the breakpoints for different numbers of bytes in a UTF-8
66: character. */
67:
68: #ifdef SUPPORT_UTF8
69:
70: const int _pcre_utf8_table1[] =
71: { 0x7f, 0x7ff, 0xffff, 0x1ffff, 0x3fffff, 0x7fffff };
72:
73: const int _pcre_utf8_table1_size = sizeof(_pcre_utf8_table1)/sizeof(int);
74:
75: /* These are the indicator bits and the mask for the data bits to set in the
76: first byte of a character, indexed by the number of additional bytes. */

```

```

77:
78: const int _pcre_utf8_table2[] = { 0, 0xc0, 0xe0, 0xf0, 0xf8, 0xfc};
79: const int _pcre_utf8_table3[] = { 0xff, 0x1f, 0x0f, 0x07, 0x03, 0x01};
80:
81: /* Table of the number of extra bytes, indexed by the first byte masked with
82: 0x3f. The highest number for a valid UTF-8 first byte is in fact 0x3d. */
83:
84: const uschar _pcre_utf8_table4[] = {
85: 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
86: 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
87: 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
88: 3,3,3,3,3,3,3,4,4,4,5,5,5,5,5 };
89:
90: /* Table to translate from particular type value to the general value. */
91:
92: const int _pcre_ucp_gentype[] = {
93: ucp_C, ucp_C, ucp_C, ucp_C, ucp_C, /* Cc, Cf, Cn, Co, Cs */
94: ucp_L, ucp_L, ucp_L, ucp_L, ucp_L, /* Ll, Lu, Lm, Lo, Lt */
95: ucp_M, ucp_M, ucp_M, /* Mc, Me, Mn */
96: ucp_N, ucp_N, ucp_N, /* Nd, Nl, No */
97: ucp_P, ucp_P, ucp_P, ucp_P, ucp_P, /* Pc, Pd, Pe, Pf, Pi */
98: ucp_P, ucp_P, /* Ps, Po */
99: ucp_S, ucp_S, ucp_S, ucp_S, /* Sc, Sk, Sm, So */
100: ucp_Z, ucp_Z, ucp_Z /* Zl, Zp, Zs */
101: };
102:
103: /* The pcre_utt[] table below translates Unicode property names into type and
104: code values. It is searched by binary chop, so must be in collating sequence of
105: name. Originally, the table contained pointers to the name strings in the first
106: field of each entry. However, that leads to a large number of relocations when
107: a shared library is dynamically loaded. A significant reduction is made by
108: putting all the names into a single, large string and then using offsets in the
109: table itself. Maintenance is more error-prone, but frequent changes to this
110: data are unlikely.
111:
112: July 2008: There is now a script called maint/GenerateUtt.py which can be used
113: to generate this data instead of maintaining it entirely by hand. */
114:
115: const char _pcre_utt_names[] =
116: "Any \0"
117: "Arabic \0"

```

118: "Armenian \0"
119: "Balinese \0"
120: "Bengali \0"
121: "Bopomofo \0"
122: "Braille \0"
123: "Buginese \0"
124: "Buhid \0"
125: "C \0"
126: "Canadian_Aboriginal \0"
127: "Carian \0"
128: "Cc \0"
129: "Cf \0"
130: "Cham \0"
131: "Cherokee \0"
132: "Cn \0"
133: "Co \0"
134: "Common \0"
135: "Coptic \0"
136: "Cs \0"
137: "Cuneiform \0"
138: "Cypriot \0"
139: "Cyrillic \0"
140: "Deseret \0"
141: "Devanagari \0"
142: "Ethiopic \0"
143: "Georgian \0"
144: "Glagolitic \0"
145: "Gothic \0"
146: "Greek \0"
147: "Gujarati \0"
148: "Gurmukhi \0"
149: "Han \0"
150: "Hangul \0"
151: "Hanunoo \0"
152: "Hebrew \0"
153: "Hiragana \0"
154: "Inherited \0"
155: "Kannada \0"
156: "Katakana \0"
157: "Kayah_Li \0"
158: "Kharoshthi \0"

159: "Khmer \0"
160: "L \0"
161: "L& \0"
162: "Lao \0"
163: "Latin \0"
164: "Lepcha \0"
165: "Limbu \0"
166: "Linear_B \0"
167: "Li \0"
168: "Lm \0"
169: "Lo \0"
170: "Lt \0"
171: "Lu \0"
172: "Lycian \0"
173: "Lydian \0"
174: "M \0"
175: "Malayalam \0"
176: "Mc \0"
177: "Me \0"
178: "Mn \0"
179: "Mongolian \0"
180: "Myanmar \0"
181: "N \0"
182: "Nd \0"
183: "New_Tai_Lue \0"
184: "Nko \0"
185: "Ni \0"
186: "No \0"
187: "Ogham \0"
188: "Ol_Chiki \0"
189: "Old_Italic \0"
190: "Old_Persian \0"
191: "Oriya \0"
192: "Osmanya \0"
193: "P \0"
194: "Pc \0"
195: "Pd \0"
196: "Pe \0"
197: "Pf \0"
198: "Phags_Pa \0"
199: "Phoenician \0"

200: "Pi \0"
 201: "Po \0"
 202: "Ps \0"
 203: "Rejang \0"
 204: "Runic \0"
 205: "S \0"
 206: "Saurashtra \0"
 207: "Sc \0"
 208: "Shavian \0"
 209: "Sinhala \0"
 210: "Sk \0"
 211: "Sm \0"
 212: "So \0"
 213: "Sundanese \0"
 214: "Syloti_Nagri \0"
 215: "Syriac \0"
 216: "Tagalog \0"
 217: "Tagbanwa \0"
 218: "Tai_Le \0"
 219: "Tamil \0"
 220: "Telugu \0"
 221: "Thaana \0"
 222: "Thai \0"
 223: "Tibetan \0"
 224: "Tifinagh \0"
 225: "Ugaritic \0"
 226: "Vai \0"
 227: "Yi \0"
 228: "Z \0"
 229: "Zl \0"
 230: "Zp \0"
 231: "Zs \0";
 232:
 233: const ucp_type_table _pcre_utt[] = {
 234: { 0, PT_ANY, 0 },
 235: { 4, PT_SC, ucp_Arabic },
 236: { 11, PT_SC, ucp_Armenian },
 237: { 20, PT_SC, ucp_Balinese },
 238: { 29, PT_SC, ucp_Bengali },
 239: { 37, PT_SC, ucp_Bopomofo },
 240: { 46, PT_SC, ucp_Braille },

241: { 54, PT_SC, ucp_Buginese },
 242: { 63, PT_SC, ucp_Buhid },
 243: { 69, PT_GC, ucp_C },
 244: { 71, PT_SC, ucp_Canadian_Aboriginal },
 245: { 91, PT_SC, ucp_Carian },
 246: { 98, PT_PC, ucp_Cc },
 247: { 101, PT_PC, ucp_Cf },
 248: { 104, PT_SC, ucp_Cham },
 249: { 109, PT_SC, ucp_Cherokee },
 250: { 118, PT_PC, ucp_Cn },
 251: { 121, PT_PC, ucp_Co },
 252: { 124, PT_SC, ucp_Common },
 253: { 131, PT_SC, ucp_Coptic },
 254: { 138, PT_PC, ucp-Cs },
 255: { 141, PT_SC, ucp_Cuneiform },
 256: { 151, PT_SC, ucp_Cypriot },
 257: { 159, PT_SC, ucp_Cyrillic },
 258: { 168, PT_SC, ucp_Deseret },
 259: { 176, PT_SC, ucp_Devanagari },
 260: { 187, PT_SC, ucp_Ethiopic },
 261: { 196, PT_SC, ucp_Georgian },
 262: { 205, PT_SC, ucp_Glagolitic },
 263: { 216, PT_SC, ucp_Gothic },
 264: { 223, PT_SC, ucp_Greek },
 265: { 229, PT_SC, ucp_Gujarati },
 266: { 238, PT_SC, ucp_Gurmukhi },
 267: { 247, PT_SC, ucp_Han },
 268: { 251, PT_SC, ucp_Hangul },
 269: { 258, PT_SC, ucp_Hanunoo },
 270: { 266, PT_SC, ucp_Hebrew },
 271: { 273, PT_SC, ucp_Hiragana },
 272: { 282, PT_SC, ucp_Inherited },
 273: { 292, PT_SC, ucp_Kannada },
 274: { 300, PT_SC, ucp_Katakana },
 275: { 309, PT_SC, ucp_Kayah_Li },
 276: { 318, PT_SC, ucp_Kharoshthi },
 277: { 329, PT_SC, ucp_Khmer },
 278: { 335, PT_GC, ucp_L },
 279: { 337, PT_LAMP, 0 },
 280: { 340, PT_SC, ucp_Lao },
 281: { 344, PT_SC, ucp_Latin },

282: { 350, PT_SC, ucp_Lepcha },
 283: { 357, PT_SC, ucp_Limbu },
 284: { 363, PT_SC, ucp_Linear_B },
 285: { 372, PT_PC, ucp_Ll },
 286: { 375, PT_PC, ucp_Lm },
 287: { 378, PT_PC, ucp_Lo },
 288: { 381, PT_PC, ucp_Lt },
 289: { 384, PT_PC, ucp_Lu },
 290: { 387, PT_SC, ucp_Lycian },
 291: { 394, PT_SC, ucp_Lydian },
 292: { 401, PT_GC, ucp_M },
 293: { 403, PT_SC, ucp_Malayalam },
 294: { 413, PT_PC, ucp_Mc },
 295: { 416, PT_PC, ucp_Me },
 296: { 419, PT_PC, ucp_Mn },
 297: { 422, PT_SC, ucp_Mongolian },
 298: { 432, PT_SC, ucp_Myanmar },
 299: { 440, PT_GC, ucp_N },
 300: { 442, PT_PC, ucp_Nd },
 301: { 445, PT_SC, ucp_New_Tai_Lue },
 302: { 457, PT_SC, ucp_Nko },
 303: { 461, PT_PC, ucp_Nl },
 304: { 464, PT_PC, ucp_No },
 305: { 467, PT_SC, ucp_Ogham },
 306: { 473, PT_SC, ucp_Ol_Chiki },
 307: { 482, PT_SC, ucp_Old_Italic },
 308: { 493, PT_SC, ucp_Old_Persian },
 309: { 505, PT_SC, ucp_Oriya },
 310: { 511, PT_SC, ucp_Osmanya },
 311: { 519, PT_GC, ucp_P },
 312: { 521, PT_PC, ucp_Pc },
 313: { 524, PT_PC, ucp_Pd },
 314: { 527, PT_PC, ucp_Pe },
 315: { 530, PT_PC, ucp_Pf },
 316: { 533, PT_SC, ucp_Phags_Pa },
 317: { 542, PT_SC, ucp_Phoenician },
 318: { 553, PT_PC, ucp_Pi },
 319: { 556, PT_PC, ucp_Po },
 320: { 559, PT_PC, ucp_Ps },
 321: { 562, PT_SC, ucp_Rejang },
 322: { 569, PT_SC, ucp_Runic },

```

323: { 575, PT_GC, ucp_S },
324: { 577, PT_SC, ucp_Saurashtra },
325: { 588, PT_PC, ucp_Sc },
326: { 591, PT_SC, ucp_Shavian },
327: { 599, PT_SC, ucp_Sinhala },
328: { 607, PT_PC, ucp_Sk },
329: { 610, PT_PC, ucp_Sm },
330: { 613, PT_PC, ucp_So },
331: { 616, PT_SC, ucp_Sundanese },
332: { 626, PT_SC, ucp_Syloti_Nagri },
333: { 639, PT_SC, ucp_Syriac },
334: { 646, PT_SC, ucp_Tagalog },
335: { 654, PT_SC, ucp_Tagbanwa },
336: { 663, PT_SC, ucp_Tai_Le },
337: { 670, PT_SC, ucp_Tamil },
338: { 676, PT_SC, ucp_Telugu },
339: { 683, PT_SC, ucp_Thaana },
340: { 690, PT_SC, ucp_Thai },
341: { 695, PT_SC, ucp_Tibetan },
342: { 703, PT_SC, ucp_Tifinagh },
343: { 712, PT_SC, ucp_Ugaritic },
344: { 721, PT_SC, ucp_Vai },
345: { 725, PT_SC, ucp_Yi },
346: { 728, PT_GC, ucp_Z },
347: { 730, PT_PC, ucp_Zl },
348: { 733, PT_PC, ucp_Zp },
349: { 736, PT_PC, ucp_Zs }
350: };
351:
352: const int _pcre_utt_size = sizeof(_pcre_utt)/sizeof(ucp_type_table);
353:
354: #endif /* SUPPORT_UTF8 */
355:
356: /* End of pcre_tables.c */

```

File: sdm/VxWorks/libRegex/pcre_get.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:         Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains some convenience functions for extracting substrings
42: from the subject string after a regex match has succeeded. The original idea
43: for these functions came from Scott Wimer. */
44:
45:
46: #ifdef HAVE_CONFIG_H
47: #include "config.h"
48: #endif
49:
50: #include "pcre_internal.h"
51:
52:
53: /*******
54: *      Find number for named string      *
55: *****/
56:
57: /* This function is used by the get_first_set() function below, as well
58: as being generally available. It assumes that names are unique.
59:
60: Arguments:
61:   code      the compiled regex
62:   stringname the name whose number is required
63:
64: Returns:    the number of the named parentheses, or a negative number
65:             (PCRE_ERROR_NOSUBSTRING) if not found
66: */
67:
68: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
69: pcre_get_stringnumber(const pcre *code, const char *stringname)
70: {
71:   int rc;
72:   int entrysize;
73:   int top, bot;
74:   uschar *nametable;
75:
76:   if ((rc = pcre_fullinfo(code, NULL, PCRE_INFO_NAMECOUNT, &top)) != 0)

```

```

77: return rc;
78: if (top <= 0) return PCRE_ERROR_NOSUBSTRING;
79:
80: if ((rc = pcre_fullinfo(code, NULL, PCRE_INFO_NAMEENTRYSIZE, &entrysize)) != 0)
81: return rc;
82: if ((rc = pcre_fullinfo(code, NULL, PCRE_INFO_NAMETABLE, &nametable)) != 0)
83: return rc;
84:
85: bot = 0;
86: while (top > bot)
87: {
88: int mid = (top + bot) / 2;
89: uschar *entry = nametable + entrysize*mid;
90: int c = strcmp(stringname, (char *)(entry + 2));
91: if (c == 0) return (entry[0] << 8) + entry[1];
92: if (c > 0) bot = mid + 1; else top = mid;
93: }
94:
95: return PCRE_ERROR_NOSUBSTRING;
96: }
97:
98:
99:
100: /*****
101: * Find (multiple) entries for named string *
102: *****/
103:
104: /* This is used by the get_first_set() function below, as well as being
105: generally available. It is used when duplicated names are permitted.
106:
107: Arguments:
108: code the compiled regex
109: stringname the name whose entries required
110: firstptr where to put the pointer to the first entry
111: lastptr where to put the pointer to the last entry
112:
113: Returns: the length of each entry, or a negative number
114: (PCRE_ERROR_NOSUBSTRING) if not found
115: */
116:
117: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION

```

```

118: pcre_get_stringtable_entries(const pcre *code, const char *stringname,
119: char **firstptr, char **lastptr)
120: {
121: int rc;
122: int entrysize;
123: int top, bot;
124: uschar *nametable, *lastentry;
125:
126: if ((rc = pcre_fullinfo(code, NULL, PCRE_INFO_NAMECOUNT, &top)) != 0)
127: return rc;
128: if (top <= 0) return PCRE_ERROR_NOSUBSTRING;
129:
130: if ((rc = pcre_fullinfo(code, NULL, PCRE_INFO_NAMEENTRYSIZE, &entrysize)) != 0)
131: return rc;
132: if ((rc = pcre_fullinfo(code, NULL, PCRE_INFO_NAMETABLE, &nametable)) != 0)
133: return rc;
134:
135: lastentry = nametable + entrysize * (top - 1);
136: bot = 0;
137: while (top > bot)
138: {
139: int mid = (top + bot) / 2;
140: uschar *entry = nametable + entrysize*mid;
141: int c = strcmp(stringname, (char *)(entry + 2));
142: if (c == 0)
143: {
144: uschar *first = entry;
145: uschar *last = entry;
146: while (first > nametable)
147: {
148: if (strcmp(stringname, (char *)(first - entrysize + 2)) != 0) break;
149: first -= entrysize;
150: }
151: while (last < lastentry)
152: {
153: if (strcmp(stringname, (char *)(last + entrysize + 2)) != 0) break;
154: last += entrysize;
155: }
156: *firstptr = (char *)first;
157: *lastptr = (char *)last;
158: return entrysize;

```

```

159: }
160: if (c > 0) bot = mid + 1; else top = mid;
161: }
162:
163: return PCRE_ERROR_NOSUBSTRING;
164: }
165:
166:
167:
168: /*****
169: *   Find first set of multiple named strings   *
170: *****/
171:
172: /* This function allows for duplicate names in the table of named substrings.
173: It returns the number of the first one that was set in a pattern match.
174:
175: Arguments:
176:   code      the compiled regex
177:   stringname the name of the capturing substring
178:   ovector    the vector of matched substrings
179:
180: Returns:     the number of the first that is set,
181:              or the number of the last one if none are set,
182:              or a negative number on error
183: */
184:
185: static int
186: get_first_set(const pcre *code, const char *stringname, int *ovector)
187: {
188: const real_pcre *re = (const real_pcre *)code;
189: int entrysize;
190: char *first, *last;
191: uschar *entry;
192: if ((re->options & PCRE_DUPNAMES) == 0 && (re->flags & PCRE_JCHANGED) == 0)
193: return pcre_get_stringnumber(code, stringname);
194: entrysize = pcre_get_stringtable_entries(code, stringname, &first, &last);
195: if (entrysize <= 0) return entrysize;
196: for (entry = (uschar *)first; entry <= (uschar *)last; entry += entrysize)
197: {
198: int n = (entry[0] << 8) + entry[1];
199: if (ovector[n*2] >= 0) return n;

```

```

200: }
201: return (first[0] << 8) + first[1];
202: }
203:
204:
205:
206:
207: /*****
208: *   Copy captured string to given buffer   *
209: *****/
210:
211: /* This function copies a single captured substring into a given buffer.
212: Note that we use memcpy() rather than strncpy() in case there are binary zeros
213: in the string.
214:
215: Arguments:
216: subject    the subject string that was matched
217: ovector    pointer to the offsets table
218: stringcount the number of substrings that were captured
219:             (i.e. the yield of the pcre_exec call, unless
220:             that was zero, in which case it should be 1/3
221:             of the offset table size)
222: stringnumber the number of the required substring
223: buffer      where to put the substring
224: size        the size of the buffer
225:
226: Returns:    if successful:
227:             the length of the copied string, not including the zero
228:             that is put on the end; can be zero
229:             if not successful:
230:             PCRE_ERROR_NOMEMORY (-6) buffer too small
231:             PCRE_ERROR_NOSUBSTRING (-7) no such captured substring
232: */
233:
234: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
235: pcre_copy_substring(const char *subject, int *ovector, int stringcount,
236: int stringnumber, char *buffer, int size)
237: {
238: int yield;
239: if (stringnumber < 0 || stringnumber >= stringcount)
240: return PCRE_ERROR_NOSUBSTRING;

```



```

241: stringnumber *= 2;
242: yield = ovector[stringnumber+1] - ovector[stringnumber];
243: if (size < yield + 1) return PCRE_ERROR_NOMEMORY;
244: memcpy(buffer, subject + ovector[stringnumber], yield);
245: buffer[yield] = 0;
246: return yield;
247: }
248:
249:
250:
251: /*****
252:  * Copy named captured string to given buffer  *
253: *****/
254:
255: /* This function copies a single captured substring into a given buffer,
256: identifying it by name. If the regex permits duplicate names, the first
257: substring that is set is chosen.
258:
259: Arguments:
260: code      the compiled regex
261: subject    the subject string that was matched
262: ovector    pointer to the offsets table
263: stringcount the number of substrings that were captured
264:            (i.e. the yield of the pcre_exec call, unless
265:            that was zero, in which case it should be 1/3
266:            of the offset table size)
267: stringname  the name of the required substring
268: buffer      where to put the substring
269: size        the size of the buffer
270:
271: Returns:    if successful:
272:            the length of the copied string, not including the zero
273:            that is put on the end; can be zero
274:            if not successful:
275:            PCRE_ERROR_NOMEMORY (-6) buffer too small
276:            PCRE_ERROR_NOSUBSTRING (-7) no such captured substring
277: */
278:
279: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
280: pcre_copy_named_substring(const pcre *code, const char *subject, int *ovector,
281: int stringcount, const char *stringname, char *buffer, int size)

```

```

282: {
283: int n = get_first_set(code, stringname, ovector);
284: if (n <= 0) return n;
285: return pcre_copy_substring(subject, ovector, stringcount, n, buffer, size);
286: }
287:
288:
289:
290: /*****
291:  *   Copy all captured strings to new store   *
292:  *****/
293:
294: /* This function gets one chunk of store and builds a list of pointers and all
295: of the captured substrings in it. A NULL pointer is put on the end of the list.
296:
297: Arguments:
298:  subject    the subject string that was matched
299:  ovector    pointer to the offsets table
300:  stringcount the number of substrings that were captured
301:             (i.e. the yield of the pcre_exec call, unless
302:             that was zero, in which case it should be 1/3
303:             of the offset table size)
304:  listptr    set to point to the list of pointers
305:
306: Returns:    if successful: 0
307:             if not successful:
308:             PCRE_ERROR_NOMEMORY (-6) failed to get store
309: */
310:
311: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
312: pcre_get_substring_list(const char *subject, int *ovector, int stringcount,
313: const char ***listptr)
314: {
315: int i;
316: int size = sizeof(char *);
317: int double_count = stringcount * 2;
318: char **stringlist;
319: char *p;
320:
321: for (i = 0; i < double_count; i += 2)
322:  size += sizeof(char *) + ovector[i+1] - ovector[i] + 1;

```

```

323:
324: stringlist = (char **)(pcre_malloc)(size);
325: if (stringlist == NULL) return PCRE_ERROR_NOMEMORY;
326:
327: *listptr = (const char **)stringlist;
328: p = (char *)(stringlist + stringcount + 1);
329:
330: for (i = 0; i < double_count; i += 2)
331: {
332:   int len = ovector[i+1] - ovector[i];
333:   memcpy(p, subject + ovector[i], len);
334:   *stringlist++ = p;
335:   p += len;
336:   *p++ = 0;
337: }
338:
339: *stringlist = NULL;
340: return 0;
341: }
342:
343:
344:
345: /*****
346:  *   Free store obtained by get_substring_list   *
347:  *****/
348:
349: /* This function exists for the benefit of people calling PCRE from non-C
350: programs that can call its functions, but not free() or (pcre_free)() directly.
351:
352: Argument:  the result of a previous pcre_get_substring_list()
353: Returns:   nothing
354: */
355:
356: PCRE_EXP_DEFN void PCRE_CALL_CONVENTION
357: pcre_free_substring_list(const char **pointer)
358: {
359:   (pcre_free)((void *)pointer);
360: }
361:
362:
363:

```

```

364: /*****
365: *   Copy captured string to new store      *
366: *****/
367:
368: /* This function copies a single captured substring into a piece of new
369: store
370:
371: Arguments:
372: subject    the subject string that was matched
373: ovector    pointer to the offsets table
374: stringcount the number of substrings that were captured
375:             (i.e. the yield of the pcre_exec call, unless
376:             that was zero, in which case it should be 1/3
377:             of the offset table size)
378: stringnumber the number of the required substring
379: stringptr   where to put a pointer to the substring
380:
381: Returns:    if successful:
382:             the length of the string, not including the zero that
383:             is put on the end; can be zero
384:             if not successful:
385:             PCRE_ERROR_NOMEMORY (-6) failed to get store
386:             PCRE_ERROR_NOSUBSTRING (-7) substring not present
387: */
388:
389: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
390: pcre_get_substring(const char *subject, int *ovector, int stringcount,
391: int stringnumber, const char **stringptr)
392: {
393: int yield;
394: char *substring;
395: if (stringnumber < 0 || stringnumber >= stringcount)
396: return PCRE_ERROR_NOSUBSTRING;
397: stringnumber *= 2;
398: yield = ovector[stringnumber+1] - ovector[stringnumber];
399: substring = (char *) (pcre_malloc)(yield + 1);
400: if (substring == NULL) return PCRE_ERROR_NOMEMORY;
401: memcpy(substring, subject + ovector[stringnumber], yield);
402: substring[yield] = 0;
403: *stringptr = substring;
404: return yield;

```

```

405: }
406:
407:
408:
409: /*****
410:  *   Copy named captured string to new store      *
411:  *****/
412:
413: /* This function copies a single captured substring, identified by name, into
414: new store. If the regex permits duplicate names, the first substring that is
415: set is chosen.
416:
417: Arguments:
418: code      the compiled regex
419: subject   the subject string that was matched
420: ovector   pointer to the offsets table
421: stringcount the number of substrings that were captured
422:           (i.e. the yield of the pcre_exec call, unless
423:           that was zero, in which case it should be 1/3
424:           of the offset table size)
425: stringname the name of the required substring
426: stringptr  where to put the pointer
427:
428: Returns:   if successful:
429:           the length of the copied string, not including the zero
430:           that is put on the end; can be zero
431:           if not successful:
432:           PCRE_ERROR_NOMEMORY (-6) couldn't get memory
433:           PCRE_ERROR_NOSUBSTRING (-7) no such captured substring
434: */
435:
436: PCRE_EXP_DEFN int PCRE_CALL_CONVENTION
437: pcre_get_named_substring(const pcre *code, const char *subject, int *ovector,
438: int stringcount, const char *stringname, const char **stringptr)
439: {
440: int n = get_first_set(code, stringname, ovector);
441: if (n <= 0) return n;
442: return pcre_get_substring(subject, ovector, stringcount, n, stringptr);
443: }
444:
445:

```

```

446:
447:
448: /*****
449: *    Free store obtained by get_substring    *
450: *****/
451:
452: /* This function exists for the benefit of people calling PCRE from non-C
453: programs that can call its functions, but not free() or (pcre_free)() directly.
454:
455: Argument:  the result of a previous pcre_get_substring()
456: Returns:  nothing
457: */
458:
459: PCRE_EXP_DEFN void PCRE_CALL_CONVENTION
460: pcre_free_substring(const char *pointer)
461: {
462: (pcre_free)((void *)pointer);
463: }
464:
465: /* End of pcre_get.c */

```

File: sdm/VxWorks/libRegex/pcre_internal.h

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5:
6: /* PCRE is a library of functions to support regular expressions whose syntax
7:  and semantics are as close as possible to those of the Perl 5 language.
8:
9:         Written by Philip Hazel
10:        Copyright (c) 1997-2008 University of Cambridge
11:
12: -----
13: Redistribution and use in source and binary forms, with or without
14: modification, are permitted provided that the following conditions are met:
15:
16:  * Redistributions of source code must retain the above copyright notice,
17:    this list of conditions and the following disclaimer.
18:
19:  * Redistributions in binary form must reproduce the above copyright
20:    notice, this list of conditions and the following disclaimer in the
21:    documentation and/or other materials provided with the distribution.
22:
23:  * Neither the name of the University of Cambridge nor the names of its
24:    contributors may be used to endorse or promote products derived from
25:    this software without specific prior written permission.
26:
27: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
28: "AS IS"
29: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
30: THE
31: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
32: PURPOSE
33: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
34: BE
35: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
36: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
37: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
38: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
39: IN
40: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
```

36: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
37: POSSIBILITY OF SUCH DAMAGE.

38: -----

39: */

40:

41: /* This header contains definitions that are shared between the different

42: modules, but which are not relevant to the exported API. This includes some

43: functions whose names all begin with "_pcre_". */

44:

45: #ifndef PCRE_INTERNAL_H

46: #define PCRE_INTERNAL_H

47:

48: /* Define DEBUG to get debugging output on stdout. */

49:

50: #if 0

51: #define DEBUG

52: #endif

53:

54: /* Use a macro for debugging printing, 'cause that eliminates the use of #ifdef

55: inline, and there are *still* stupid compilers about that don't like indented

56: pre-processor statements, or at least there were when I first wrote this. After

57: all, it had only been about 10 years then...

58:

59: It turns out that the Mac Debugging.h header also defines the macro DPRINTF, so

60: be absolutely sure we get our version. */

61:

62: #undef DPRINTF

63: #ifdef DEBUG

64: #define DPRINTF(p) printf p

65: #else

66: #define DPRINTF(p) /* Nothing */

67: #endif

68:

69:

70: /* Standard C headers plus the external interface definition. The only time

71: setjmp and stdarg are used is when NO_RECURSE is set. */

72:

73: #include <ctype.h>

74: #include <limits.h>

75: #include <setjmp.h>

76: #include <stdarg.h>


```

77: #include <stddef.h>
78: #include <stdio.h>
79: #include <stdlib.h>
80: #include <string.h>
81:
82: /* When compiling a DLL for Windows, the exported symbols have to be declared
83: using some MS magic. I found some useful information on this web page:
84: http://msdn2.microsoft.com/en-us/library/y4h7bcy6\(VS.80\).aspx. According to the
85: information there, using __declspec(dllexport) without "extern" we have a
86: definition; with "extern" we have a declaration. The settings here override the
87: setting in pcre.h (which is included below); it defines only PCRE_EXP_DECL,
88: which is all that is needed for applications (they just import the symbols). We
89: use:
90:
91: PCRE_EXP_DECL    for declarations
92: PCRE_EXP_DEFN    for definitions of exported functions
93: PCRE_EXP_DATA_DEFN for definitions of exported variables
94:
95: The reason for the two DEFN macros is that in non-Windows environments, one
96: does not want to have "extern" before variable definitions because it leads to
97: compiler warnings. So we distinguish between functions and variables. In
98: Windows, the two should always be the same.
99:
100: The reason for wrapping this in #ifndef PCRE_EXP_DECL is so that pcretest,
101: which is an application, but needs to import this file in order to "peek" at
102: internals, can #include pcre.h first to get an application's-eye view.
103:
104: In principle, people compiling for non-Windows, non-Unix-like (i.e. uncommon,
105: special-purpose environments) might want to stick other stuff in front of
106: exported symbols. That's why, in the non-Windows case, we set PCRE_EXP_DEFN and
107: PCRE_EXP_DATA_DEFN only if they are not already set. */
108:
109: #ifndef PCRE_EXP_DECL
110: #  ifdef _WIN32
111: #    ifndef PCRE_STATIC
112: #      define PCRE_EXP_DECL    extern __declspec(dllexport)
113: #      define PCRE_EXP_DEFN    __declspec(dllexport)
114: #      define PCRE_EXP_DATA_DEFN __declspec(dllexport)
115: #    else
116: #      define PCRE_EXP_DECL    extern
117: #      define PCRE_EXP_DEFN

```

```

118: #   define PCRE_EXP_DATA_DEFN
119: #   endif
120: #   else
121: #   ifdef __cplusplus
122: #       define PCRE_EXP_DECL    extern "C"
123: #       else
124: #       define PCRE_EXP_DECL    extern
125: #       endif
126: #   ifndef PCRE_EXP_DEFN
127: #       define PCRE_EXP_DEFN    PCRE_EXP_DECL
128: #       endif
129: #   ifndef PCRE_EXP_DATA_DEFN
130: #       define PCRE_EXP_DATA_DEFN
131: #       endif
132: #   endif
133: #endif
134:
135: /* When compiling with the MSVC compiler, it is sometimes necessary to include
136: a "calling convention" before exported function names. (This is secondhand
137: information; I know nothing about MSVC myself). For example, something like
138:
139: void __cdecl function(...)
140:
141: might be needed. In order so make this easy, all the exported functions have
142: PCRE_CALL_CONVENTION just before their names. It is rarely needed; if not
143: set, we ensure here that it has no effect. */
144:
145: #ifndef PCRE_CALL_CONVENTION
146: #define PCRE_CALL_CONVENTION
147: #endif
148:
149: /* We need to have types that specify unsigned 16-bit and 32-bit integers. We
150: cannot determine these outside the compilation (e.g. by running a program as
151: part of "configure") because PCRE is often cross-compiled for use on other
152: systems. Instead we make use of the maximum sizes that are available at
153: preprocessor time in standard C environments. */
154:
155: #if USHRT_MAX == 65535
156: typedef unsigned short pcre_uint16;
157: typedef short pcre_int16;
158: #elif UINT_MAX == 65535

```

```

159: typedef unsigned int pcre_uint16;
160: typedef int pcre_int16;
161: #else
162: #error Cannot determine a type for 16-bit unsigned integers
163: #endif
164:
165: #if UINT_MAX == 4294967295
166: typedef unsigned int pcre_uint32;
167: typedef int pcre_int32;
168: #elif ULONG_MAX == 4294967295
169: typedef unsigned long int pcre_uint32;
170: typedef long int pcre_int32;
171: #else
172: #error Cannot determine a type for 32-bit unsigned integers
173: #endif
174:
175: /* All character handling must be done as unsigned characters. Otherwise there
176: are problems with top-bit-set characters and functions such as isspace().
177: However, we leave the interface to the outside world as char *, because that
178: should make things easier for callers. We define a short type for unsigned char
179: to save lots of typing. I tried "uchar", but it causes problems on Digital
180: Unix, where it is defined in sys/types, so use "uschar" instead. */
181:
182: typedef unsigned char uschar;
183:
184: /* This is an unsigned int value that no character can ever have. UTF-8
185: characters only go up to 0x7fffffff (though Unicode doesn't go beyond
186: 0x0010ffff). */
187:
188: #define NOTACHAR 0xffffffff
189:
190: /* PCRE is able to support several different kinds of newline (CR, LF, CRLF,
191: "any" and "anycrlf" at present). The following macros are used to package up
192: testing for newlines. NLBLOCK, PSSTART, and PSEND are defined in the various
193: modules to indicate in which datablock the parameters exist, and what the
194: start/end of string field names are. */
195:
196: #define NLTYPE_FIXED 0 /* Newline is a fixed length string */
197: #define NLTYPE_ANY 1 /* Newline is any Unicode line ending */
198: #define NLTYPE_ANYCRLF 2 /* Newline is CR, LF, or CRLF */
199:

```

```

200: /* This macro checks for a newline at the given position */
201:
202: #define IS_NEWLINE(p) \
203: ((NLBLOCK->nlttype != NLTYPE_FIXED)? \
204: ((p) < NLBLOCK->PSEND && \
205:  _pcre_is_newline((p), NLBLOCK->nlttype, NLBLOCK->PSEND, &(NLBLOCK->nllen), \
206:   utf8)) \
207: : \
208: ((p) <= NLBLOCK->PSEND - NLBLOCK->nllen && \
209:  (p)[0] == NLBLOCK->nl[0] && \
210:  (NLBLOCK->nllen == 1 || (p)[1] == NLBLOCK->nl[1]) \
211:  ) \
212: )
213:
214: /* This macro checks for a newline immediately preceding the given position */
215:
216: #define WAS_NEWLINE(p) \
217: ((NLBLOCK->nlttype != NLTYPE_FIXED)? \
218: ((p) > NLBLOCK->PSSTART && \
219:  _pcre_was_newline((p), NLBLOCK->nlttype, NLBLOCK->PSSTART, \
220:   &(NLBLOCK->nllen), utf8)) \
221: : \
222: ((p) >= NLBLOCK->PSSTART + NLBLOCK->nllen && \
223:  (p)[-NLBLOCK->nllen] == NLBLOCK->nl[0] && \
224:  (NLBLOCK->nllen == 1 || (p)[-NLBLOCK->nllen+1] == NLBLOCK->nl[1]) \
225:  ) \
226: )
227:
228: /* When PCRE is compiled as a C++ library, the subject pointer can be replaced
229: with a custom type. This makes it possible, for example, to allow pcre_exec()
230: to process subject strings that are discontinuous by using a smart pointer
231: class. It must always be possible to inspect all of the subject string in
232: pcre_exec() because of the way it backtracks. Two macros are required in the
233: normal case, for sign-unspecified and unsigned char pointers. The former is
234: used for the external interface and appears in pcre.h, which is why its name
235: must begin with PCRE_. */
236:
237: #ifdef CUSTOM_SUBJECT_PTR
238: #define PCRE_SPTR CUSTOM_SUBJECT_PTR
239: #define USPTR CUSTOM_SUBJECT_PTR
240: #else

```

```

241: #define PCRE_SPTR const char *
242: #define USPTR const unsigned char *
243: #endif
244:
245:
246:
247: /* Include the public PCRE header and the definitions of UCP character property
248: values. */
249:
250: #include "pcre.h"
251: #include "ucp.h"
252:
253: /* When compiling for use with the Virtual Pascal compiler, these functions
254: need to have their names changed. PCRE must be compiled with the -DVPCOMPAT
255: option on the command line. */
256:
257: #ifdef VPCOMPAT
258: #define strlen(s)    _strlen(s)
259: #define strncmp(s1,s2,m) _strncmp(s1,s2,m)
260: #define memcmp(s,c,n)  _memcmp(s,c,n)
261: #define memcpy(d,s,n)  _memcpy(d,s,n)
262: #define memmove(d,s,n) _memmove(d,s,n)
263: #define memset(s,c,n)  _memset(s,c,n)
264: #else /* VPCOMPAT */
265:
266: /* To cope with SunOS4 and other systems that lack memmove() but have bcopy(),
267: define a macro for memmove() if HAVE_MEMMOVE is false, provided that HAVE_BCOPY
268: is set. Otherwise, include an emulating function for those systems that have
269: neither (there some non-Unix environments where this is the case). */
270:
271: #ifndef HAVE_MEMMOVE
272: #undef memmove      /* some systems may have a macro */
273: #ifdef HAVE_BCOPY
274: #define memmove(a, b, c) bcopy(b, a, c)
275: #else /* HAVE_BCOPY */
276: static void *
277: pcre_memmove(void *d, const void *s, size_t n)
278: {
279: size_t i;
280: unsigned char *dest = (unsigned char *)d;
281: const unsigned char *src = (const unsigned char *)s;

```

```

282: if (dest > src)
283: {
284:     dest += n;
285:     src += n;
286:     for (i = 0; i < n; ++i) *--dest = *--src;
287:     return (void *)dest;
288: }
289: else
290: {
291:     for (i = 0; i < n; ++i) *dest++ = *src++;
292:     return (void *) (dest - n);
293: }
294: }
295: #define memmove(a, b, c) pcre_memmove(a, b, c)
296: #endif /* not HAVE_BCOPY */
297: #endif /* not HAVE_MEMMOVE */
298: #endif /* not VPCOMPAT */
299:
300:
301: /* PCRE keeps offsets in its compiled code as 2-byte quantities (always stored
302: in big-endian order) by default. These are used, for example, to link from the
303: start of a subpattern to its alternatives and its end. The use of 2 bytes per
304: offset limits the size of the compiled regex to around 64K, which is big enough
305: for almost everybody. However, I received a request for an even bigger limit.
306: For this reason, and also to make the code easier to maintain, the storing and
307: loading of offsets from the byte string is now handled by the macros that are
308: defined here.
309:
310: The macros are controlled by the value of LINK_SIZE. This defaults to 2 in
311: the config.h file, but can be overridden by using -D on the command line. This
312: is automated on Unix systems via the "configure" command. */
313:
314: #if LINK_SIZE == 2
315:
316: #define PUT(a,n,d) \
317:   (a[n] = (d) >> 8), \
318:   (a[(n)+1] = (d) & 255)
319:
320: #define GET(a,n) \
321:   (((a)[n] << 8) | (a)[(n)+1])
322:

```

```

323: #define MAX_PATTERN_SIZE (1 << 16)
324:
325:
326: #elif LINK_SIZE == 3
327:
328: #define PUT(a,n,d)    \
329:  (a[n] = (d) >> 16),  \
330:  (a[(n)+1] = (d) >> 8), \
331:  (a[(n)+2] = (d) & 255)
332:
333: #define GET(a,n) \
334:  (((a)[n] << 16) | ((a)[(n)+1] << 8) | (a)[(n)+2])
335:
336: #define MAX_PATTERN_SIZE (1 << 24)
337:
338:
339: #elif LINK_SIZE == 4
340:
341: #define PUT(a,n,d)    \
342:  (a[n] = (d) >> 24),  \
343:  (a[(n)+1] = (d) >> 16), \
344:  (a[(n)+2] = (d) >> 8),  \
345:  (a[(n)+3] = (d) & 255)
346:
347: #define GET(a,n) \
348:  (((a)[n] << 24) | ((a)[(n)+1] << 16) | ((a)[(n)+2] << 8) | (a)[(n)+3])
349:
350: #define MAX_PATTERN_SIZE (1 << 30) /* Keep it positive */
351:
352:
353: #else
354: #error LINK_SIZE must be either 2, 3, or 4
355: #endif
356:
357:
358: /* Convenience macro defined in terms of the others */
359:
360: #define PUTINC(a,n,d)  PUT(a,n,d), a += LINK_SIZE
361:
362:
363: /* PCRE uses some other 2-byte quantities that do not change when the size of

```

```

364: offsets changes. There are used for repeat counts and for other things such as
365: capturing parenthesis numbers in back references. */
366:
367: #define PUT2(a,n,d) \
368: a[n] = (d) >> 8; \
369: a[(n)+1] = (d) & 255
370:
371: #define GET2(a,n) \
372: (((a)[n] << 8) | (a)[(n)+1])
373:
374: #define PUT2INC(a,n,d) PUT2(a,n,d), a += 2
375:
376:
377: /* When UTF-8 encoding is being used, a character is no longer just a single
378: byte. The macros for character handling generate simple sequences when used in
379: byte-mode, and more complicated ones for UTF-8 characters. BACKCHAR should
380: never be called in byte mode. To make sure it can never even appear when UTF-8
381: support is omitted, we don't even define it. */
382:
383: #ifndef SUPPORT_UTF8
384: #define GETCHAR(c, eptr) c = *eptr;
385: #define GETCHARTEST(c, eptr) c = *eptr;
386: #define GETCHARINC(c, eptr) c = *eptr++;
387: #define GETCHARINCTEST(c, eptr) c = *eptr++;
388: #define GETCHARLEN(c, eptr, len) c = *eptr;
389: /* #define BACKCHAR(eptr) */
390:
391: #else /* SUPPORT_UTF8 */
392:
393: /* Get the next UTF-8 character, not advancing the pointer. This is called when
394: we know we are in UTF-8 mode. */
395:
396: #define GETCHAR(c, eptr) \
397: c = *eptr; \
398: if (c >= 0xc0) \
399: { \
400: int gcii; \
401: int gcaa = _pcre_utf8_table4[c & 0x3f]; /* Number of additional bytes */ \
402: int gcss = 6*gcaa; \
403: c = (c & _pcre_utf8_table3[gcaa]) << gcss; \
404: for (gcii = 1; gcii <= gcaa; gcii++) \

```



```

405:     { \
406:     gcss -= 6; \
407:     c |= (eptr[gcii] & 0x3f) << gcss; \
408:     } \
409: }
410:
411: /* Get the next UTF-8 character, testing for UTF-8 mode, and not advancing the
412: pointer. */
413:
414: #define GETCHARTEST(c, eptr) \
415: c = *eptr; \
416: if (utf8 && c >= 0xc0) \
417: { \
418:     int gcii; \
419:     int gcaa = _pcre_utf8_table4[c & 0x3f]; /* Number of additional bytes */ \
420:     int gcss = 6*gcaa; \
421:     c = (c & _pcre_utf8_table3[gcaa]) << gcss; \
422:     for (gcii = 1; gcii <= gcaa; gcii++) \
423:     { \
424:         gcss -= 6; \
425:         c |= (eptr[gcii] & 0x3f) << gcss; \
426:     } \
427: }
428:
429: /* Get the next UTF-8 character, advancing the pointer. This is called when we
430: know we are in UTF-8 mode. */
431:
432: #define GETCHARINC(c, eptr) \
433: c = *eptr++; \
434: if (c >= 0xc0) \
435: { \
436:     int gcaa = _pcre_utf8_table4[c & 0x3f]; /* Number of additional bytes */ \
437:     int gcss = 6*gcaa; \
438:     c = (c & _pcre_utf8_table3[gcaa]) << gcss; \
439:     while (gcaa-- > 0) \
440:     { \
441:         gcss -= 6; \
442:         c |= (*eptr++ & 0x3f) << gcss; \
443:     } \
444: }
445:

```

```

446: /* Get the next character, testing for UTF-8 mode, and advancing the pointer */
447:
448: #define GETCHARINCTEST(c, eptr) \
449:   c = *eptr++; \
450:   if (utf8 && c >= 0xc0) \
451:   { \
452:     int gcaa = _pcre_utf8_table4[c & 0x3f]; /* Number of additional bytes */ \
453:     int gcss = 6*gcaa; \
454:     c = (c & _pcre_utf8_table3[gcaa]) << gcss; \
455:     while (gcaa-- > 0) \
456:     { \
457:       gcss -= 6; \
458:       c |= (*eptr++ & 0x3f) << gcss; \
459:     } \
460:   }
461:
462: /* Get the next UTF-8 character, not advancing the pointer, incrementing length
463: if there are extra bytes. This is called when we know we are in UTF-8 mode. */
464:
465: #define GETCHARLEN(c, eptr, len) \
466:   c = *eptr; \
467:   if (c >= 0xc0) \
468:   { \
469:     int gcii; \
470:     int gcaa = _pcre_utf8_table4[c & 0x3f]; /* Number of additional bytes */ \
471:     int gcss = 6*gcaa; \
472:     c = (c & _pcre_utf8_table3[gcaa]) << gcss; \
473:     for (gcii = 1; gcii <= gcaa; gcii++) \
474:     { \
475:       gcss -= 6; \
476:       c |= (eptr[gcii] & 0x3f) << gcss; \
477:     } \
478:     len += gcaa; \
479:   }
480:
481: /* If the pointer is not at the start of a character, move it back until
482: it is. This is called only in UTF-8 mode - we don't put a test within the macro
483: because almost all calls are already within a block of UTF-8 only code. */
484:
485: #define BACKCHAR(eptr) while((*eptr & 0xc0) == 0x80) eptr--
486:

```

```

487: #endif
488:
489:
490: /* In case there is no definition of offsetof() provided - though any proper
491: Standard C system should have one. */
492:
493: #ifndef offsetof
494: #define offsetof(p_type,field) ((size_t)&(((p_type *)0)->field))
495: #endif
496:
497:
498: /* These are the public options that can change during matching. */
499:
500: #define PCRE_IMS (PCRE_CASELESS|PCRE_MULTILINE|PCRE_DOTALL)
501:
502: /* Private flags containing information about the compiled regex. They used to
503: live at the top end of the options word, but that got almost full, so now they
504: are in a 16-bit flags word. */
505:
506: #define PCRE_NOPARTIAL    0x0001 /* can't use partial with this regex */
507: #define PCRE_FIRSTSET    0x0002 /* first_byte is set */
508: #define PCRE_REQCHSET    0x0004 /* req_byte is set */
509: #define PCRE_STARTLINE    0x0008 /* start after \n for multiline */
510: #define PCRE_JCHANGED    0x0010 /* j option used in regex */
511: #define PCRE_HASCORLRF    0x0020 /* explicit \r or \n in pattern */
512:
513: /* Options for the "extra" block produced by pcre_study(). */
514:
515: #define PCRE_STUDY_MAPPED 0x01 /* a map of starting chars exists */
516:
517: /* Masks for identifying the public options that are permitted at compile
518: time, run time, or study time, respectively. */
519:
520: #define PCRE_NEWLINE_BITS
(PCRE_NEWLINE_CR|PCRE_NEWLINE_LF|PCRE_NEWLINE_ANY| \
521: PCRE_NEWLINE_ANYCRLF)
522:
523: #define PUBLIC_OPTIONS \
524: (PCRE_CASELESS|PCRE_EXTENDED|PCRE_ANCHORED|PCRE_MULTILINE| \
525: PCRE_DOTALL|PCRE_DOLLAR_ENDONLY|PCRE_EXTRA|PCRE_UNGREEDY|PCRE_UTF8| \

```

```

526: PCRE_NO_AUTO_CAPTURE|PCRE_NO_UTF8_CHECK|PCRE_AUTO_CALLOUT|PCRE_FIRSTLI
NE| \
527: PCRE_DUPNAMES|PCRE_NEWLINE_BITS|PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE|
\
528: PCRE_JAVASCRIPT_COMPAT)
529:
530: #define PUBLIC_EXEC_OPTIONS \
531: (PCRE_ANCHORED|PCRE_NOTBOL|PCRE_NOTEOL|PCRE_NOTEMPTY|PCRE_NO_UTF8_CHE
CK| \
532: PCRE_PARTIAL|PCRE_NEWLINE_BITS|PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE)
533:
534: #define PUBLIC_DFA_EXEC_OPTIONS \
535: (PCRE_ANCHORED|PCRE_NOTBOL|PCRE_NOTEOL|PCRE_NOTEMPTY|PCRE_NO_UTF8_CHE
CK| \
536: PCRE_PARTIAL|PCRE_DFA_SHORTEST|PCRE_DFA_RESTART|PCRE_NEWLINE_BITS| \
537: PCRE_BSR_ANYCRLF|PCRE_BSR_UNICODE)
538:
539: #define PUBLIC_STUDY_OPTIONS 0 /* None defined */
540:
541: /* Magic number to provide a small check against being handed junk. Also used
542: to detect whether a pattern was compiled on a host of different endianness. */
543:
544: #define MAGIC_NUMBER 0x50435245UL /* 'PCRE' */
545:
546: /* Negative values for the firstchar and reqchar variables */
547:
548: #define REQ_UNSET (-2)
549: #define REQ_NONE (-1)
550:
551: /* The maximum remaining length of subject we are prepared to search for a
552: req_byte match. */
553:
554: #define REQ_BYTE_MAX 1000
555:
556: /* Flags added to firstbyte or reqbyte; a "non-literal" item is either a
557: variable-length repeat, or a anything other than literal characters. */
558:
559: #define REQ_CASELESS 0x0100 /* indicates caselessness */
560: #define REQ_VARY 0x0200 /* reqbyte followed non-literal item */

```

```

561:
562: /* Miscellaneous definitions. The #ifndef is to pacify compiler warnings in
563: environments where these macros are defined elsewhere. */
564:
565: #ifndef FALSE
566: typedef int BOOL;
567:
568: #define FALSE 0
569: #define TRUE 1
570: #endif
571:
572: /* Escape items that are just an encoding of a particular data value. */
573:
574: #ifndef ESC_e
575: #define ESC_e 27
576: #endif
577:
578: #ifndef ESC_f
579: #define ESC_f '\f'
580: #endif
581:
582: #ifndef ESC_n
583: #define ESC_n '\n'
584: #endif
585:
586: #ifndef ESC_r
587: #define ESC_r '\r'
588: #endif
589:
590: /* We can't officially use ESC_t because it is a POSIX reserved identifier
591: (presumably because of all the others like size_t). */
592:
593: #ifndef ESC_tee
594: #define ESC_tee '\t'
595: #endif
596:
597: /* Codes for different types of Unicode property */
598:
599: #define PT_ANY 0 /* Any property - matches all chars */
600: #define PT_LAMP 1 /* L& - the union of Lu, Ll, Lt */
601: #define PT_GC 2 /* General characteristic (e.g. L) */

```

```

602: #define PT_PC      3  /* Particular characteristic (e.g. Lu) */
603: #define PT_SC      4  /* Script (e.g. Han) */
604:
605: /* Flag bits and data types for the extended class (OP_XCLASS) for classes that
606: contain UTF-8 characters with values greater than 255. */
607:
608: #define XCL_NOT    0x01  /* Flag: this is a negative class */
609: #define XCL_MAP    0x02  /* Flag: a 32-byte map is present */
610:
611: #define XCL_END     0  /* Marks end of individual items */
612: #define XCL_SINGLE  1  /* Single item (one multibyte char) follows */
613: #define XCL_RANGE   2  /* A range (two multibyte chars) follows */
614: #define XCL_PROP    3  /* Unicode property (2-byte property code follows) */
615: #define XCL_NOTPROP 4  /* Unicode inverted property (ditto) */
616:
617: /* These are escaped items that aren't just an encoding of a particular data
618: value such as \n. They must have non-zero values, as check_escape() returns
619: their negation. Also, they must appear in the same order as in the opcode
620: definitions below, up to ESC_z. There's a dummy for OP_ANY because it
621: corresponds to "." rather than an escape sequence, and another for OP_ALLANY
622: (which is used for [^] in JavaScript compatibility mode).
623:
624: The final escape must be ESC_REF as subsequent values are used for
625: backreferences ( \1, \2, \3, etc). There are two tests in the code for an escape
626: greater than ESC_b and less than ESC_Z to detect the types that may be
627: repeated. These are the types that consume characters. If any new escapes are
628: put in between that don't consume a character, that code will have to change.
629: */
630:
631: enum { ESC_A = 1, ESC_G, ESC_K, ESC_B, ESC_b, ESC_D, ESC_d, ESC_S, ESC_s,
632:       ESC_W, ESC_w, ESC_dum1, ESC_dum2, ESC_C, ESC_P, ESC_p, ESC_R, ESC_H,
633:       ESC_h, ESC_V, ESC_v, ESC_X, ESC_Z, ESC_z, ESC_E, ESC_Q, ESC_g, ESC_k,
634:       ESC_REF };
635:
636:
637: /* Opcode table: Starting from 1 (i.e. after OP_END), the values up to
638: OP_EOD must correspond in order to the list of escapes immediately above.
639:
640: *** NOTE NOTE NOTE *** Whenever this list is updated, the two macro definitions
641: that follow must also be updated to match. There is also a table called
642: "coptable" in pcre_dfa_exec.c that must be updated. */

```

```

643:
644: enum {
645:   OP_END,          /* 0 End of pattern */
646:
647:   /* Values corresponding to backslashed metacharacters */
648:
649:   OP_SOD,          /* 1 Start of data: \A */
650:   OP_SOM,          /* 2 Start of match (subject + offset): \G */
651:   OP_SET_SOM,      /* 3 Set start of match (\K) */
652:   OP_NOT_WORD_BOUNDARY, /* 4 \B */
653:   OP_WORD_BOUNDARY, /* 5 \b */
654:   OP_NOT_DIGIT,    /* 6 \D */
655:   OP_DIGIT,        /* 7 \d */
656:   OP_NOT_WHITESPACE, /* 8 \S */
657:   OP_WHITESPACE,   /* 9 \s */
658:   OP_NOT_WORDCHAR, /* 10 \W */
659:   OP_WORDCHAR,     /* 11 \w */
660:   OP_ANY,          /* 12 Match any character (subject to DOTALL) */
661:   OP_ALLANY,       /* 13 Match any character (not subject to DOTALL) */
662:   OP_ANYBYTE,      /* 14 Match any byte (\C); different to OP_ANY for UTF-8 */
663:   OP_NOTPROP,      /* 15 \P (not Unicode property) */
664:   OP_PROP,         /* 16 \p (Unicode property) */
665:   OP_ANYNL,        /* 17 \R (any newline sequence) */
666:   OP_NOT_HSPACE,   /* 18 \H (not horizontal whitespace) */
667:   OP_HSPACE,       /* 19 \h (horizontal whitespace) */
668:   OP_NOT_VSPACE,   /* 20 \V (not vertical whitespace) */
669:   OP_VSPACE,       /* 21 \v (vertical whitespace) */
670:   OP_EXTUNI,       /* 22 \X (extended Unicode sequence) */
671:   OP_EODN,         /* 23 End of data or \n at end of data: \Z. */
672:   OP_EOD,          /* 24 End of data: \z */
673:
674:   OP_OPT,          /* 25 Set runtime options */
675:   OP_CIRC,         /* 26 Start of line - varies with multiline switch */
676:   OP_DOLL,         /* 27 End of line - varies with multiline switch */
677:   OP_CHAR,         /* 28 Match one character, casefully */
678:   OP_CHARNC,       /* 29 Match one character, caselessly */
679:   OP_NOT,          /* 30 Match one character, not the following one */
680:
681:   OP_STAR,         /* 31 The maximizing and minimizing versions of */
682:   OP_MINSTAR,      /* 32 these six opcodes must come in pairs, with */
683:   OP_PLUS,         /* 33 the minimizing one second. */

```

684: OP_MINPLUS, /* 34 This first set applies to single characters.*/
685: OP_QUERY, /* 35 */
686: OP_MINQUERY, /* 36 */
687:
688: OP_UPTO, /* 37 From 0 to n matches */
689: OP_MINUPTO, /* 38 */
690: OP_EXACT, /* 39 Exactly n matches */
691:
692: OP_POSSTAR, /* 40 Possessified star */
693: OP_POSPLUS, /* 41 Possessified plus */
694: OP_POSQUERY, /* 42 Possessified query */
695: OP_POSUPTO, /* 43 Possessified upto */
696:
697: OP_NOTSTAR, /* 44 The maximizing and minimizing versions of */
698: OP_NOTMINSTAR, /* 45 these six opcodes must come in pairs, with */
699: OP_NOTPLUS, /* 46 the minimizing one second. They must be in */
700: OP_NOTMINPLUS, /* 47 exactly the same order as those above. */
701: OP_NOTQUERY, /* 48 This set applies to "not" single characters. */
702: OP_NOTMINQUERY, /* 49 */
703:
704: OP_NOTUPTO, /* 50 From 0 to n matches */
705: OP_NOTMINUPTO, /* 51 */
706: OP_NOTEXACT, /* 52 Exactly n matches */
707:
708: OP_NOTPOSSTAR, /* 53 Possessified versions */
709: OP_NOTPOSPLUS, /* 54 */
710: OP_NOTPOSQUERY, /* 55 */
711: OP_NOTPOSUPTO, /* 56 */
712:
713: OP_TYPESTAR, /* 57 The maximizing and minimizing versions of */
714: OP_TYPEMINSTAR, /* 58 these six opcodes must come in pairs, with */
715: OP_TYPEPLUS, /* 59 the minimizing one second. These codes must */
716: OP_TYPEMINPLUS, /* 60 be in exactly the same order as those above. */
717: OP_TYPEQUERY, /* 61 This set applies to character types such as \d */
718: OP_TYPEMINQUERY, /* 62 */
719:
720: OP_TYPEUPTO, /* 63 From 0 to n matches */
721: OP_TYPEMINUPTO, /* 64 */
722: OP_TYPEEXACT, /* 65 Exactly n matches */
723:
724: OP_TYPEPOSSTAR, /* 66 Possessified versions */


```

725: OP_TYPEPOSPLUS, /* 67 */
726: OP_TYPEPOSQUERY, /* 68 */
727: OP_TYPEPOSUPTO, /* 69 */
728:
729: OP_CRSTAR, /* 70 The maximizing and minimizing versions of */
730: OP_CRMINSTAR, /* 71 all these opcodes must come in pairs, with */
731: OP_CRPLUS, /* 72 the minimizing one second. These codes must */
732: OP_CRMINPLUS, /* 73 be in exactly the same order as those above. */
733: OP_CRQUERY, /* 74 These are for character classes and back refs */
734: OP_CRMINQUERY, /* 75 */
735: OP_CRRANGE, /* 76 These are different to the three sets above. */
736: OP_CRMINRANGE, /* 77 */
737:
738: OP_CLASS, /* 78 Match a character class, chars < 256 only */
739: OP_NCLASS, /* 79 Same, but the bitmap was created from a negative
740: class - the difference is relevant only when a UTF-8
741: character > 255 is encountered. */
742:
743: OP_XCLASS, /* 80 Extended class for handling UTF-8 chars within the
744: class. This does both positive and negative. */
745:
746: OP_REF, /* 81 Match a back reference */
747: OP_RECURSE, /* 82 Match a numbered subpattern (possibly recursive) */
748: OP_CALLOUT, /* 83 Call out to external function if provided */
749:
750: OP_ALT, /* 84 Start of alternation */
751: OP_KET, /* 85 End of group that doesn't have an unbounded repeat */
752: OP_KETRMX, /* 86 These two must remain together and in this */
753: OP_KETRMN, /* 87 order. They are for groups the repeat for ever. */
754:
755: /* The assertions must come before BRA, CBRA, ONCE, and COND.*/
756:
757: OP_ASSERT, /* 88 Positive lookahead */
758: OP_ASSERT_NOT, /* 89 Negative lookahead */
759: OP_ASSERTBACK, /* 90 Positive lookbehind */
760: OP_ASSERTBACK_NOT, /* 91 Negative lookbehind */
761: OP_REVERSE, /* 92 Move pointer back - used in lookbehind assertions */
762:
763: /* ONCE, BRA, CBRA, and COND must come after the assertions, with ONCE first,
764: as there's a test for >= ONCE for a subpattern that isn't an assertion. */
765:

```

```

766: OP_ONCE,      /* 93 Atomic group */
767: OP_BRA,       /* 94 Start of non-capturing bracket */
768: OP_CBRA,      /* 95 Start of capturing bracket */
769: OP_COND,      /* 96 Conditional group */
770:
771: /* These three must follow the previous three, in the same order. There's a
772: check for >= SBRA to distinguish the two sets. */
773:
774: OP_SBRA,       /* 97 Start of non-capturing bracket, check empty */
775: OP_SCBRA,      /* 98 Start of capturing bracket, check empty */
776: OP_SCOND,      /* 99 Conditional group, check empty */
777:
778: OP_CREF,       /* 100 Used to hold a capture number as condition */
779: OP_RREF,       /* 101 Used to hold a recursion number as condition */
780: OP_DEF,        /* 102 The DEFINE condition */
781:
782: OP_BRAZERO,    /* 103 These two must remain together and in this */
783: OP_BRAMINZERO, /* 104 order. */
784:
785: /* These are backtracking control verbs */
786:
787: OP_PRUNE,      /* 105 */
788: OP_SKIP,       /* 106 */
789: OP_THEN,       /* 107 */
790: OP_COMMIT,     /* 108 */
791:
792: /* These are forced failure and success verbs */
793:
794: OP_FAIL,       /* 109 */
795: OP_ACCEPT,     /* 110 */
796:
797: /* This is used to skip a subpattern with a {0} quantifier */
798:
799: OP_SKIPZERO    /* 111 */
800: };
801:
802:
803: /* This macro defines textual names for all the opcodes. These are used only
804: for debugging. The macro is referenced only in pcre_printint.c. */
805:
806: #define OP_NAME_LIST \

```

```

807: "End", "\\A", "\\G", "\\K", "\\B", "\\b", "\\D", "\\d",      \
808: "\\S", "\\s", "\\W", "\\w", "Any", "AllAny", "Anybyte",    \
809: "notprop", "prop", "\\R", "\\H", "\\h", "\\V", "\\v",      \
810: "extuni", "\\Z", "\\z",                                     \
811: "Opt", "^", "$", "char", "charnc", "not",                 \
812: "*", "*?", "+", "+?", "?", "??", "{", "{", "{",           \
813: "+", "+", "?+", "{",                                       \
814: "*", "*?", "+", "+?", "?", "??", "{", "{", "{",           \
815: "+", "+", "?+", "{",                                       \
816: "*", "*?", "+", "+?", "?", "??", "{", "{", "{",           \
817: "+", "+", "?+", "{",                                       \
818: "*", "*?", "+", "+?", "?", "??", "{", "{",               \
819: "class", "nclass", "xclass", "Ref", "Recurse", "Callout",  \
820: "Alt", "Ket", "KetRmax", "KetRmin", "Assert", "Assert not", \
821: "AssertB", "AssertB not", "Reverse",                       \
822: "Once", "Bra", "CBra", "Cond", "SBra", "SCBra", "SCond",  \
823: "Cond ref", "Cond rec", "Cond def", "Brazero", "Braminzero", \
824: "*PRUNE", "*SKIP", "*THEN", "*COMMIT", "*FAIL", "*ACCEPT", \
825: "Skip zero"
826:
827:
828: /* This macro defines the length of fixed length operations in the compiled
829: regex. The lengths are used when searching for specific things, and also in the
830: debugging printing of a compiled regex. We use a macro so that it can be
831: defined close to the definitions of the opcodes themselves.
832:
833: As things have been extended, some of these are no longer fixed lengths, but are
834: minima instead. For example, the length of a single-character repeat may vary
835: in UTF-8 mode. The code that uses this table must know about such things. */
836:
837: #define OP_LENGTHS \
838: 1, /* End */ \
839: 1, 1, 1, 1, 1, /* \A, \G, \K, \B, \b */ \
840: 1, 1, 1, 1, 1, 1, /* \D, \d, \S, \s, \W, \w */ \
841: 1, 1, 1, /* Any, AllAny, Anybyte */ \
842: 3, 3, 1, /* NOTPROP, PROP, EXTUNI */ \
843: 1, 1, 1, 1, 1, /* \R, \H, \h, \V, \v */ \
844: 1, 1, 2, 1, 1, /* \Z, \z, Opt, ^, $ */ \
845: 2, /* Char - the minimum length */ \
846: 2, /* Charnc - the minimum length */ \
847: 2, /* not */ \

```

```

848: /* Positive single-char repeats          ** These are */ \
849: 2, 2, 2, 2, 2, 2,          /* *, *?, +, +?, ?, ??  ** minima in */ \
850: 4, 4, 4,                  /* upto, minupto, exact  ** UTF-8 mode */ \
851: 2, 2, 2, 4,              /* *+, ++, ?+, upto+          */ \
852: /* Negative single-char repeats - only for chars < 256          */ \
853: 2, 2, 2, 2, 2, 2,          /* NOT *, *?, +, +?, ?, ??          */ \
854: 4, 4, 4,                  /* NOT upto, minupto, exact          */ \
855: 2, 2, 2, 4,              /* Possessive *, +, ?, upto          */ \
856: /* Positive type repeats                      */ \
857: 2, 2, 2, 2, 2, 2,          /* Type *, *?, +, +?, ?, ??          */ \
858: 4, 4, 4,                  /* Type upto, minupto, exact          */ \
859: 2, 2, 2, 4,              /* Possessive *+, ++, ?+, upto+          */ \
860: /* Character class & ref repeats                      */ \
861: 1, 1, 1, 1, 1, 1,          /* *, *?, +, +?, ?, ??          */ \
862: 5, 5,                    /* CRRANGE, CRMINRANGE          */ \
863: 33,                      /* CLASS                      */ \
864: 33,                      /* NCLASS                      */ \
865: 0,                       /* XCLASS - variable length          */ \
866: 3,                       /* REF                      */ \
867: 1+LINK_SIZE,            /* RECURSE                      */ \
868: 2+2*LINK_SIZE,          /* CALLOUT                      */ \
869: 1+LINK_SIZE,            /* Alt                      */ \
870: 1+LINK_SIZE,            /* Ket                      */ \
871: 1+LINK_SIZE,            /* KetRmax                      */ \
872: 1+LINK_SIZE,            /* KetRmin                      */ \
873: 1+LINK_SIZE,            /* Assert                      */ \
874: 1+LINK_SIZE,            /* Assert not                  */ \
875: 1+LINK_SIZE,            /* Assert behind              */ \
876: 1+LINK_SIZE,            /* Assert behind not          */ \
877: 1+LINK_SIZE,            /* Reverse                      */ \
878: 1+LINK_SIZE,            /* ONCE                      */ \
879: 1+LINK_SIZE,            /* BRA                      */ \
880: 3+LINK_SIZE,            /* CBRA                      */ \
881: 1+LINK_SIZE,            /* COND                      */ \
882: 1+LINK_SIZE,            /* SBRA                      */ \
883: 3+LINK_SIZE,            /* SCBRA                      */ \
884: 1+LINK_SIZE,            /* SCOND                      */ \
885: 3,                      /* CREF                      */ \
886: 3,                      /* RREF                      */ \
887: 1,                      /* DEF                      */ \
888: 1, 1,                  /* BRAZERO, BRAMINZERO          */ \

```

```

889: 1, 1, 1, 1,          /* PRUNE, SKIP, THEN, COMMIT,      */ \
890: 1, 1, 1             /* FAIL, ACCEPT, SKIPZERO        */
891:
892:
893: /* A magic value for OP_RREF to indicate the "any recursion" condition. */
894:
895: #define RREF_ANY 0xffff
896:
897: /* Error code numbers. They are given names so that they can more easily be
898: tracked. */
899:
900: enum { ERR0, ERR1, ERR2, ERR3, ERR4, ERR5, ERR6, ERR7, ERR8, ERR9,
901:      ERR10, ERR11, ERR12, ERR13, ERR14, ERR15, ERR16, ERR17, ERR18, ERR19,
902:      ERR20, ERR21, ERR22, ERR23, ERR24, ERR25, ERR26, ERR27, ERR28, ERR29,
903:      ERR30, ERR31, ERR32, ERR33, ERR34, ERR35, ERR36, ERR37, ERR38, ERR39,
904:      ERR40, ERR41, ERR42, ERR43, ERR44, ERR45, ERR46, ERR47, ERR48, ERR49,
905:      ERR50, ERR51, ERR52, ERR53, ERR54, ERR55, ERR56, ERR57, ERR58, ERR59,
906:      ERR60, ERR61, ERR62, ERR63, ERR64 };
907:
908: /* The real format of the start of the pcre block; the index of names and the
909: code vector run on as long as necessary after the end. We store an explicit
910: offset to the name table so that if a regex is compiled on one host, saved, and
911: then run on another where the size of pointers is different, all might still
912: be well. For the case of compiled-on-4 and run-on-8, we include an extra
913: pointer that is always NULL. For future-proofing, a few dummy fields were
914: originally included - even though you can never get this planning right - but
915: there is only one left now.
916:
917: NOTE NOTE NOTE:
918: Because people can now save and re-use compiled patterns, any additions to this
919: structure should be made at the end, and something earlier (e.g. a new
920: flag in the options or one of the dummy fields) should indicate that the new
921: fields are present. Currently PCRE always sets the dummy fields to zero.
922: NOTE NOTE NOTE:
923: */
924:
925: typedef struct real_pcre {
926:   pcre_uint32 magic_number;
927:   pcre_uint32 size;          /* Total that was malloced */
928:   pcre_uint32 options;       /* Public options */
929:   pcre_uint16 flags;         /* Private flags */

```

```

930: pcre_uint16 dummy1;      /* For future use */
931: pcre_uint16 top_bracket;
932: pcre_uint16 top_backref;
933: pcre_uint16 first_byte;
934: pcre_uint16 req_byte;
935: pcre_uint16 name_table_offset; /* Offset to name table that follows */
936: pcre_uint16 name_entry_size; /* Size of any name items */
937: pcre_uint16 name_count;      /* Number of name items */
938: pcre_uint16 ref_count;      /* Reference count */
939:
940: const unsigned char *tables; /* Pointer to tables or NULL for std */
941: const unsigned char *nullpad; /* NULL padding */
942: } real_pcre;
943:
944: /* The format of the block used to store data from pcre_study(). The same
945: remark (see NOTE above) about extending this structure applies. */
946:
947: typedef struct pcre_study_data {
948:     pcre_uint32 size;      /* Total that was malloced */
949:     pcre_uint32 options;
950:     uschar start_bits[32];
951: } pcre_study_data;
952:
953: /* Structure for passing "static" information around between the functions
954: doing the compiling, so that they are thread-safe. */
955:
956: typedef struct compile_data {
957:     const uschar *lcc;      /* Points to lower casing table */
958:     const uschar *fcc;      /* Points to case-flipping table */
959:     const uschar *cbits;    /* Points to character type table */
960:     const uschar *ctypes;    /* Points to table of type maps */
961:     const uschar *start_workspace; /* The start of working space */
962:     const uschar *start_code; /* The start of the compiled code */
963:     const uschar *start_pattern; /* The start of the pattern */
964:     const uschar *end_pattern; /* The end of the pattern */
965:     uschar *hwm;            /* High watermark of workspace */
966:     uschar *name_table;     /* The name/number table */
967:     int names_found;        /* Number of entries so far */
968:     int name_entry_size;    /* Size of each entry */
969:     int bracount;          /* Count of capturing parens as we compile */
970:     int final_bracount;     /* Saved value after first pass */

```

```

971: int top_backref;          /* Maximum back reference */
972: unsigned int backref_map; /* Bitmap of low back refs */
973: int external_options;     /* External (initial) options */
974: int external_flags;       /* External flag bits to be set */
975: int req_varyopt;          /* "After variable item" flag for reqbyte */
976: BOOL had_accept;          /* (*ACCEPT) encountered */
977: int nltype;               /* Newline type */
978: int nllen;                /* Newline string length */
979: uschar nl[4];             /* Newline string when fixed length */
980: } compile_data;
981:
982: /* Structure for maintaining a chain of pointers to the currently incomplete
983: branches, for testing for left recursion. */
984:
985: typedef struct branch_chain {
986: struct branch_chain *outer;
987: uschar *current;
988: } branch_chain;
989:
990: /* Structure for items in a linked list that represents an explicit recursive
991: call within the pattern. */
992:
993: typedef struct recursion_info {
994: struct recursion_info *prevrec; /* Previous recursion record (or NULL) */
995: int group_num;                 /* Number of group that was called */
996: const uschar *after_call;      /* "Return value": points after the call in the expr */
997: USPTR save_start;              /* Old value of mstart */
998: int *offset_save;              /* Pointer to start of saved offsets */
999: int saved_max;                 /* Number of saved offsets */
1000: } recursion_info;
1001:
1002: /* Structure for building a chain of data for holding the values of the subject
1003: pointer at the start of each subpattern, so as to detect when an empty string
1004: has been matched by a subpattern - to break infinite loops. */
1005:
1006: typedef struct eptrblock {
1007: struct eptrblock *epb_prev;
1008: USPTR epb_saved_eptr;
1009: } eptrblock;
1010:
1011:

```

```

1012: /* Structure for passing "static" information around between the functions
1013: doing traditional NFA matching, so that they are thread-safe. */
1014:
1015: typedef struct match_data {
1016:     unsigned long int match_call_count;    /* As it says */
1017:     unsigned long int match_limit;        /* As it says */
1018:     unsigned long int match_limit_recursion; /* As it says */
1019:     int *offset_vector;    /* Offset vector */
1020:     int  offset_end;       /* One past the end */
1021:     int  offset_max;       /* The maximum usable for return data */
1022:     int  nltype;           /* Newline type */
1023:     int  nllen;            /* Newline string length */
1024:     uschar nl[4];         /* Newline string when fixed */
1025:     const uschar *lcc;     /* Points to lower casing table */
1026:     const uschar *ctypes;  /* Points to table of type maps */
1027:     BOOL  offset_overflow; /* Set if too many extractions */
1028:     BOOL  notbol;          /* NOTBOL flag */
1029:     BOOL  noteol;          /* NOTEOL flag */
1030:     BOOL  utf8;            /* UTF8 flag */
1031:     BOOL  jscript_compat;  /* JAVASCRIPT_COMPAT flag */
1032:     BOOL  endonly;         /* Dollar not before final \n */
1033:     BOOL  notempty;        /* Empty string match not wanted */
1034:     BOOL  partial;         /* PARTIAL flag */
1035:     BOOL  hitend;          /* Hit the end of the subject at some point */
1036:     BOOL  bsr_anycrlf;     /* \R is just any CRLF, not full Unicode */
1037:     const uschar *start_code; /* For use when recursing */
1038:     USPTR start_subject;    /* Start of the subject string */
1039:     USPTR end_subject;      /* End of the subject string */
1040:     USPTR start_match_ptr;  /* Start of matched string */
1041:     USPTR end_match_ptr;    /* Subject position at end match */
1042:     int  end_offset_top;    /* Highwater mark at end of match */
1043:     int  capture_last;      /* Most recent capture number */
1044:     int  start_offset;      /* The start offset value */
1045:     eptrblock *eptrchain;   /* Chain of eptrblocks for tail recursions */
1046:     int  eptrn;             /* Next free eptrblock */
1047:     recursion_info *recursive; /* Linked list of recursion data */
1048:     void *callout_data;     /* To pass back to callouts */
1049: } match_data;
1050:
1051: /* A similar structure is used for the same purpose by the DFA matching
1052: functions. */

```



```

1053:
1054: typedef struct dfa_match_data {
1055:     const uschar *start_code; /* Start of the compiled pattern */
1056:     const uschar *start_subject; /* Start of the subject string */
1057:     const uschar *end_subject; /* End of subject string */
1058:     const uschar *tables; /* Character tables */
1059:     int moptions; /* Match options */
1060:     int poptions; /* Pattern options */
1061:     int nltype; /* Newline type */
1062:     int nllen; /* Newline string length */
1063:     uschar nl[4]; /* Newline string when fixed */
1064:     void *callout_data; /* To pass back to callouts */
1065: } dfa_match_data;
1066:
1067: /* Bit definitions for entries in the pcre_ctype table. */
1068:
1069: #define ctype_space 0x01
1070: #define ctype_letter 0x02
1071: #define ctype_digit 0x04
1072: #define ctype_xdigit 0x08
1073: #define ctype_word 0x10 /* alphanumeric or '_' */
1074: #define ctype_meta 0x80 /* regexp meta char or zero (end pattern) */
1075:
1076: /* Offsets for the bitmap tables in pcre_cbits. Each table contains a set
1077: of bits for a class map. Some classes are built by combining these tables. */
1078:
1079: #define cbit_space 0 /* [:space:] or \s */
1080: #define cbit_xdigit 32 /* [:xdigit:] */
1081: #define cbit_digit 64 /* [:digit:] or \d */
1082: #define cbit_upper 96 /* [:upper:] */
1083: #define cbit_lower 128 /* [:lower:] */
1084: #define cbit_word 160 /* [:word:] or \w */
1085: #define cbit_graph 192 /* [:graph:] */
1086: #define cbit_print 224 /* [:print:] */
1087: #define cbit_punct 256 /* [:punct:] */
1088: #define cbit_cntrl 288 /* [:cntrl:] */
1089: #define cbit_length 320 /* Length of the cbits table */
1090:
1091: /* Offsets of the various tables from the base tables pointer, and
1092: total length. */
1093:

```

```

1094: #define lcc_offset    0
1095: #define fcc_offset  256
1096: #define cbits_offset 512
1097: #define ctypes_offset (cbits_offset + cbit_length)
1098: #define tables_length (ctypes_offset + 256)
1099:
1100: /* Layout of the UCP type table that translates property names into types and
1101: codes. Each entry used to point directly to a name, but to reduce the number of
1102: relocations in shared libraries, it now has an offset into a single string
1103: instead. */
1104:
1105: typedef struct {
1106:     pcre_uint16 name_offset;
1107:     pcre_uint16 type;
1108:     pcre_uint16 value;
1109: } ucp_type_table;
1110:
1111:
1112: /* Internal shared data tables. These are tables that are used by more than one
1113: of the exported public functions. They have to be "external" in the C sense,
1114: but are not part of the PCRE public API. The data for these tables is in the
1115: pcre_tables.c module. */
1116:
1117: extern const int  _pcre_utf8_table1[];
1118: extern const int  _pcre_utf8_table2[];
1119: extern const int  _pcre_utf8_table3[];
1120: extern const uschar _pcre_utf8_table4[];
1121:
1122: extern const int  _pcre_utf8_table1_size;
1123:
1124: extern const char  _pcre_utt_names[];
1125: extern const ucp_type_table _pcre_utt[];
1126: extern const int  _pcre_utt_size;
1127:
1128: extern const uschar _pcre_default_tables[];
1129:
1130: extern const uschar _pcre_OP_lengths[];
1131:
1132:
1133: /* Internal shared functions. These are functions that are used by more than
1134: one of the exported public functions. They have to be "external" in the C

```

```

1135: sense, but are not part of the PCRE public API. */
1136:
1137: extern BOOL      _pcre_is_newline(const uschar *, int, const uschar *,
1138:      int *, BOOL);
1139: extern int      _pcre_ord2utf8(int, uschar *);
1140: extern real_pcre *_pcre_try_flipped(const real_pcre *, real_pcre *,
1141:      const pcre_study_data *, pcre_study_data *);
1142: extern int      _pcre_valid_utf8(const uschar *, int);
1143: extern BOOL      _pcre_was_newline(const uschar *, int, const uschar *,
1144:      int *, BOOL);
1145: extern BOOL      _pcre_xclass(int, const uschar *);
1146:
1147:
1148: /* Unicode character database (UCD) */
1149:
1150: typedef struct {
1151:  uschar script;
1152:  uschar chartype;
1153:  pcre_int32 other_case;
1154: } ucd_record;
1155:
1156: extern const ucd_record _pcre_uct_records[];
1157: extern const uschar _pcre_uct_stage1[];
1158: extern const pcre_uint16 _pcre_uct_stage2[];
1159: extern const int _pcre_uct_gentype[];
1160:
1161:
1162: /* UCD access macros */
1163:
1164: #define UCD_BLOCK_SIZE 128
1165: #define GET_UCD(ch) (_pcre_uct_records + \
1166:      _pcre_uct_stage2[_pcre_uct_stage1[(ch) / UCD_BLOCK_SIZE] * \
1167:      UCD_BLOCK_SIZE + ch % UCD_BLOCK_SIZE])
1168:
1169: #define UCD_CHARTYPE(ch) GET_UCD(ch)->chartype
1170: #define UCD_SCRIPT(ch) GET_UCD(ch)->script
1171: #define UCD_CATEGORY(ch) _pcre_uct_gentype[UCD_CHARTYPE(ch)]
1172: #define UCD_OTHERCASE(ch) (ch + GET_UCD(ch)->other_case)
1173:
1174: #endif
1175:

```

1176: /* End of pcre_internal.h */

File: sdm/VxWorks/libRegex/pcre_xclass.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains an internal function that is used to match an extended
42: class (one that contains characters whose values are > 255). It is used by both
43: pcre_exec() and pcre_def_exec(). */
44:
45:
46: #ifdef HAVE_CONFIG_H
47: #include "config.h"
48: #endif
49:
50: #include "pcre_internal.h"
51:
52:
53: /*****
54:  *      Match character against an XCLASS      *
55:  *****/
56:
57: /* This function is called to match a character against an extended class that
58: might contain values > 255.
59:
60: Arguments:
61:   c      the character
62:   data   points to the flag byte of the XCLASS data
63:
64: Returns:  TRUE if character matches, else FALSE
65: */
66:
67: BOOL
68: _pcre_xclass(int c, const uschar *data)
69: {
70:   int t;
71:   BOOL negated = (*data & XCL_NOT) != 0;
72:
73:   /* Character values < 256 are matched against a bitmap, if one is present. If
74:   not, we still carry on, because there may be ranges that start below 256 in the
75:   additional data. */
76:

```

```

77: if (c < 256)
78: {
79: if ((*data & XCL_MAP) != 0 && (data[1 + c/8] & (1 << (c&7))) != 0)
80: return !negated; /* char found */
81: }
82:
83: /* First skip the bit map if present. Then match against the list of Unicode
84: properties or large chars or ranges that end with a large char. We won't ever
85: encounter XCL_PROP or XCL_NOTPROP when UCP support is not compiled. */
86:
87: if ((*data++ & XCL_MAP) != 0) data += 32;
88:
89: while ((t = *data++) != XCL_END)
90: {
91: int x, y;
92: if (t == XCL_SINGLE)
93: {
94: GETCHARINC(x, data);
95: if (c == x) return !negated;
96: }
97: else if (t == XCL_RANGE)
98: {
99: GETCHARINC(x, data);
100: GETCHARINC(y, data);
101: if (c >= x && c <= y) return !negated;
102: }
103:
104: #ifdef SUPPORT_UCP
105: else /* XCL_PROP & XCL_NOTPROP */
106: {
107: const ucd_record * prop = GET_UCD(c);
108:
109: switch(*data)
110: {
111: case PT_ANY:
112: if (t == XCL_PROP) return !negated;
113: break;
114:
115: case PT_LAMP:
116: if ((prop->chartype == ucp_Lu || prop->chartype == ucp_Ll || prop->chartype == ucp_Lt) ==
117: (t == XCL_PROP)) return !negated;

```

```

118:    break;
119:
120:    case PT_GC:
121:        if ((data[1] == _pcre_ucp_gentype[prop->chartype]) == (t == XCL_PROP)) return !negated;
122:        break;
123:
124:    case PT_PC:
125:        if ((data[1] == prop->chartype) == (t == XCL_PROP)) return !negated;
126:        break;
127:
128:    case PT_SC:
129:        if ((data[1] == prop->script) == (t == XCL_PROP)) return !negated;
130:        break;
131:
132:    /* This should never occur, but compilers may mutter if there is no
133:       default. */
134:
135:    default:
136:        return FALSE;
137:    }
138:
139:    data += 2;
140: }
141: #endif /* SUPPORT_UCP */
142: }
143:
144: return negated; /* char did not match */
145: }
146:
147: /* End of pcre_xclass.c */

```


File: sdm/VxWorks/libRegex/pcre_valid_utf8.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3:  *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains an internal function for validating UTF-8 character
42: strings. */
43:
44:
45: #ifdef HAVE_CONFIG_H
46: #include "config.h"
47: #endif
48:
49: #include "pcre_internal.h"
50:
51:
52: /*****
53:  *      Validate a UTF-8 string      *
54:  *****/
55:
56: /* This function is called (optionally) at the start of compile or match, to
57: validate that a supposed UTF-8 string is actually valid. The early check means
58: that subsequent code can assume it is dealing with a valid string. The check
59: can be turned off for maximum performance, but the consequences of supplying
60: an invalid string are then undefined.
61:
62: Originally, this function checked according to RFC 2279, allowing for values in
63: the range 0 to 0x7fffffff, up to 6 bytes long, but ensuring that they were in
64: the canonical format. Once somebody had pointed out RFC 3629 to me (it
65: obsoletes 2279), additional restrictions were applied. The values are now
66: limited to be between 0 and 0x0010ffff, no more than 4 bytes long, and the
67: subrange 0xd000 to 0xdfff is excluded.
68:
69: Arguments:
70:  string    points to the string
71:  length    length of string, or -1 if the string is zero-terminated
72:
73: Returns:   < 0  if the string is a valid UTF-8 string
74:            >= 0 otherwise; the value is the offset of the bad byte
75: */
76:

```

```

77: int
78: _pcre_valid_utf8(const uschar *string, int length)
79: {
80: #ifdef SUPPORT_UTF8
81: register const uschar *p;
82:
83: if (length < 0)
84: {
85: for (p = string; *p != 0; p++);
86: length = p - string;
87: }
88:
89: for (p = string; length-- > 0; p++)
90: {
91: register int ab;
92: register int c = *p;
93: if (c < 128) continue;
94: if (c < 0xc0) return p - string;
95: ab = _pcre_utf8_table4[c & 0x3f]; /* Number of additional bytes */
96: if (length < ab || ab > 3) return p - string;
97: length -= ab;
98:
99: /* Check top bits in the second byte */
100: if ((*++p & 0xc0) != 0x80) return p - string;
101:
102: /* Check for overlong sequences for each different length, and for the
103: excluded range 0xd000 to 0xdfff. */
104:
105: switch (ab)
106: {
107: /* Check for xx00 000x (overlong sequence) */
108:
109: case 1:
110: if ((c & 0x3e) == 0) return p - string;
111: continue; /* We know there aren't any more bytes to check */
112:
113: /* Check for 1110 0000, xx0x xxxx (overlong sequence) or
114: 1110 1101, 1010 xxxx (0xd000 - 0xdfff) */
115:
116: case 2:
117: if ((c == 0xe0 && (*p & 0x20) == 0) ||

```

```

118:     (c == 0xed && *p >= 0xa0))
119:     return p - string;
120:     break;
121:
122:     /* Check for 1111 0000, xx00 xxxx (overlong sequence) or
123:        greater than 0x0010ffff (f4 8f bf bf) */
124:
125:     case 3:
126:     if ((c == 0xf0 && (*p & 0x30) == 0) ||
127:         (c > 0xf4) ||
128:         (c == 0xf4 && *p > 0x8f))
129:         return p - string;
130:     break;
131:
132: #if 0
133:     /* These cases can no longer occur, as we restrict to a maximum of four
134:        bytes nowadays. Leave the code here in case we ever want to add an option
135:        for longer sequences. */
136:
137:     /* Check for 1111 1000, xx00 0xxx */
138:     case 4:
139:     if (c == 0xf8 && (*p & 0x38) == 0) return p - string;
140:     break;
141:
142:     /* Check for leading 0xfe or 0xff, and then for 1111 1100, xx00 00xx */
143:     case 5:
144:     if (c == 0xfe || c == 0xff ||
145:         (c == 0xfc && (*p & 0x3c) == 0)) return p - string;
146:     break;
147: #endif
148:
149:     }
150:
151:     /* Check for valid bytes after the 2nd, if any; all must start 10 */
152:     while (--ab > 0)
153:     {
154:         if ((*++p) & 0xc0) != 0x80) return p - string;
155:     }
156: }
157: #else
158: (void)(string); /* Keep picky compilers happy */

```

```
159: (void)(length);
160: #endif
161:
162: return -1;
163: }
164:
165: /* End of pcre_valid_utf8.c */
```

File: sdm/VxWorks/libRegex/config.h

```
1: /* config.h. Generated from config.h.in by configure. */
2: /* config.h.in. Generated from configure.ac by autoheader. */
3:
4:
5: /* On Unix-like systems config.h.in is converted by "configure" into config.h.
6: Some other environments also support the use of "configure". PCRE is written in
7: Standard C, but there are a few non-standard things it can cope with, allowing
8: it to run on SunOS4 and other "close to standard" systems.
9:
10: If you are going to build PCRE "by hand" on a system without "configure" you
11: should copy the distributed config.h.generic to config.h, and then set up the
12: macro definitions the way you need them. You must then add -DHAVE_CONFIG_H to
13: all of your compile commands, so that config.h is included at the start of
14: every source.
15:
16: Alternatively, you can avoid editing by using -D on the compiler command line
17: to set the macro values. In this case, you do not have to set -DHAVE_CONFIG_H.
18:
19: PCRE uses memmove() if HAVE_MEMMOVE is set to 1; otherwise it uses bcopy() if
20: HAVE_BCOPY is set to 1. If your system has neither bcopy() nor memmove(), set
21: them both to 0; an emulation function will be used. */
22:
23: /* By default, the \R escape sequence matches any Unicode line ending
24: character or sequence of characters. If BSR_ANYCRLF is defined, this is
25: changed so that backslash-R matches only CR, LF, or CRLF. The build-time
26: default can be overridden by the user of PCRE at runtime. On systems that
27: support it, "configure" can be used to override the default. */
28: /* #undef BSR_ANYCRLF */
29:
30: /* If you are compiling for a system that uses EBCDIC instead of ASCII
31: character codes, define this macro as 1. On systems that can use
32: "configure", this can be done via --enable-ebcdic. */
33: /* #undef EBCDIC */
34:
35: /* Define to 1 if you have the `bcopy' function. */
36: #ifndef HAVE_BCOPY
37: #define HAVE_BCOPY 1
38: #endif
39:
```

```

40: /* Define to 1 if you have the <bits/type_traits.h> header file. */
41: /* #undef HAVE_BITS_TYPE_TRAITS_H */
42:
43: /* Define to 1 if you have the <bzlib.h> header file. */
44: #ifndef HAVE_BZLIB_H
45: #define HAVE_BZLIB_H 1
46: #endif
47:
48: /* Define to 1 if you have the <dirent.h> header file. */
49: #ifndef HAVE_DIRENT_H
50: #define HAVE_DIRENT_H 1
51: #endif
52:
53: /* Define to 1 if you have the <dlfcn.h> header file. */
54: #ifndef HAVE_DLFCN_H
55: #define HAVE_DLFCN_H 1
56: #endif
57:
58: /* Define to 1 if you have the <inttypes.h> header file. */
59: #ifndef HAVE_INTTYPES_H
60: #define HAVE_INTTYPES_H 1
61: #endif
62:
63: /* Define to 1 if you have the <limits.h> header file. */
64: #ifndef HAVE_LIMITS_H
65: #define HAVE_LIMITS_H 1
66: #endif
67:
68: /* Define to 1 if the system has the type `long long'. */
69: #ifndef HAVE_LONG_LONG
70: #define HAVE_LONG_LONG 1
71: #endif
72:
73: /* Define to 1 if you have the `memmove' function. */
74: #ifndef HAVE_MEMMOVE
75: #define HAVE_MEMMOVE 1
76: #endif
77:
78: /* Define to 1 if you have the <memory.h> header file. */
79: #ifndef HAVE_MEMORY_H
80: #define HAVE_MEMORY_H 1

```

```

81: #endif
82:
83: /* Define to 1 if you have the <readline/history.h> header file. */
84: #ifndef HAVE_READLINE_HISTORY_H
85: #define HAVE_READLINE_HISTORY_H 1
86: #endif
87:
88: /* Define to 1 if you have the <readline/readline.h> header file. */
89: #ifndef HAVE_READLINE_READLINE_H
90: #define HAVE_READLINE_READLINE_H 1
91: #endif
92:
93: /* Define to 1 if you have the <stdint.h> header file. */
94: #ifndef HAVE_STDINT_H
95: #define HAVE_STDINT_H 1
96: #endif
97:
98: /* Define to 1 if you have the <stdlib.h> header file. */
99: #ifndef HAVE_STDLIB_H
100: #define HAVE_STDLIB_H 1
101: #endif
102:
103: /* Define to 1 if you have the `strerror' function. */
104: #ifndef HAVE_STRERROR
105: #define HAVE_STRERROR 1
106: #endif
107:
108: /* Define to 1 if you have the <string> header file. */
109: #ifndef HAVE_STRING
110: #define HAVE_STRING 1
111: #endif
112:
113: /* Define to 1 if you have the <strings.h> header file. */
114: #ifndef HAVE_STRINGS_H
115: #define HAVE_STRINGS_H 1
116: #endif
117:
118: /* Define to 1 if you have the <string.h> header file. */
119: #ifndef HAVE_STRING_H
120: #define HAVE_STRING_H 1
121: #endif

```



```

122:
123: /* Define to 1 if you have the `strtoll' function. */
124: /* #undef HAVE_STRTOLL */
125:
126: /* Define to 1 if you have the `strtoq' function. */
127: #ifndef HAVE_STRTOQ
128: #define HAVE_STRTOQ 1
129: #endif
130:
131: /* Define to 1 if you have the <sys/stat.h> header file. */
132: #ifndef HAVE_SYS_STAT_H
133: #define HAVE_SYS_STAT_H 1
134: #endif
135:
136: /* Define to 1 if you have the <sys/types.h> header file. */
137: #ifndef HAVE_SYS_TYPES_H
138: #define HAVE_SYS_TYPES_H 1
139: #endif
140:
141: /* Define to 1 if you have the <type_traits.h> header file. */
142: /* #undef HAVE_TYPE_TRAITS_H */
143:
144: /* Define to 1 if you have the <unistd.h> header file. */
145: #ifndef HAVE_UNISTD_H
146: #define HAVE_UNISTD_H 1
147: #endif
148:
149: /* Define to 1 if the system has the type `unsigned long long'. */
150: #ifndef HAVE_UNSIGNED_LONG_LONG
151: #define HAVE_UNSIGNED_LONG_LONG 1
152: #endif
153:
154: /* Define to 1 if you have the <windows.h> header file. */
155: /* #undef HAVE_WINDOWS_H */
156:
157: /* Define to 1 if you have the <zlib.h> header file. */
158: #ifndef HAVE_ZLIB_H
159: #define HAVE_ZLIB_H 1
160: #endif
161:
162: /* Define to 1 if you have the `_strtoi64' function. */

```

```

163: /* #undef HAVE__STRTOI64 */
164:
165: /* The value of LINK_SIZE determines the number of bytes used to store links
166:  as offsets within the compiled regex. The default is 2, which allows for
167:  compiled patterns up to 64K long. This covers the vast majority of cases.
168:  However, PCRE can also be compiled to use 3 or 4 bytes instead. This allows
169:  for longer patterns in extreme cases. On systems that support it,
170:  "configure" can be used to override this default. */
171: #ifndef LINK_SIZE
172: #define LINK_SIZE 2
173: #endif
174:
175: /* The value of MATCH_LIMIT determines the default number of times the
176:  internal match() function can be called during a single execution of
177:  pcre_exec(). There is a runtime interface for setting a different limit.
178:  The limit exists in order to catch runaway regular expressions that take
179:  for ever to determine that they do not match. The default is set very large
180:  so that it does not accidentally catch legitimate cases. On systems that
181:  support it, "configure" can be used to override this default default. */
182: #ifndef MATCH_LIMIT
183: #define MATCH_LIMIT 10000000
184: #endif
185:
186: /* The above limit applies to all calls of match(), whether or not they
187:  increase the recursion depth. In some environments it is desirable to limit
188:  the depth of recursive calls of match() more strictly, in order to restrict
189:  the maximum amount of stack (or heap, if NO_RECURSE is defined) that is
190:  used. The value of MATCH_LIMIT_RECURSION applies only to recursive calls of
191:  match(). To have any useful effect, it must be less than the value of
192:  MATCH_LIMIT. The default is to use the same value as MATCH_LIMIT. There is
193:  a runtime method for setting a different limit. On systems that support it,
194:  "configure" can be used to override the default. */
195: #ifndef MATCH_LIMIT_RECURSION
196: #define MATCH_LIMIT_RECURSION MATCH_LIMIT
197: #endif
198:
199: /* This limit is parameterized just in case anybody ever wants to change it.
200:  Care must be taken if it is increased, because it guards against integer
201:  overflow caused by enormously large patterns. */
202: #ifndef MAX_NAME_COUNT
203: #define MAX_NAME_COUNT 10000

```

```

204: #endif
205:
206: /* This limit is parameterized just in case anybody ever wants to change it.
207:  Care must be taken if it is increased, because it guards against integer
208:  overflow caused by enormously large patterns. */
209: #ifndef MAX_NAME_SIZE
210: #define MAX_NAME_SIZE 32
211: #endif
212:
213: /* The value of NEWLINE determines the newline character sequence. On systems
214:  that support it, "configure" can be used to override the default, which is
215:  10. The possible values are 10 (LF), 13 (CR), 3338 (CRLF), -1 (ANY), or -2
216:  (ANYCRLF). */
217: #ifndef NEWLINE
218: #define NEWLINE 10
219: #endif
220:
221: /* PCRE uses recursive function calls to handle backtracking while matching.
222:  This can sometimes be a problem on systems that have stacks of limited
223:  size. Define NO_RECURSE to get a version that doesn't use recursion in the
224:  match() function; instead it creates its own stack by steam using
225:  pcre_recurse_malloc() to obtain memory from the heap. For more detail, see
226:  the comments and other stuff just above the match() function. On systems
227:  that support it, "configure" can be used to set this in the Makefile (use
228:  --disable-stack-for-recursion). */
229: /* #undef NO_RECURSE */
230:
231: /* Name of package */
232: #define PACKAGE "pcre"
233:
234: /* Define to the address where bug reports for this package should be sent. */
235: #define PACKAGE_BUGREPORT ""
236:
237: /* Define to the full name of this package. */
238: #define PACKAGE_NAME "PCRE"
239:
240: /* Define to the full name and version of this package. */
241: #define PACKAGE_STRING "PCRE 7.8"
242:
243: /* Define to the one symbol short name of this package. */
244: #define PACKAGE_TARNAME "pcre"

```

```

245:
246: /* Define to the version of this package. */
247: #define PACKAGE_VERSION "7.8"
248:
249:
250: /* If you are compiling for a system other than a Unix-like system or
251:  Win32, and it needs some magic to be inserted before the definition
252:  of a function that is exported by the library, define this macro to
253:  contain the relevant magic. If you do not define this macro, it
254:  defaults to "extern" for a C compiler and "extern C" for a C++
255:  compiler on non-Win32 systems. This macro appears at the start of
256:  every exported function that is part of the external API. It does
257:  not appear on functions that are "external" in the C sense, but
258:  which are internal to the library. */
259: /* #undef PCRE_EXP_DEFN */
260:
261: /* Define if linking statically (TODO: make nice with Libtool) */
262: /* #undef PCRE_STATIC */
263:
264: /* When calling PCRE via the POSIX interface, additional working storage is
265:  required for holding the pointers to capturing substrings because PCRE
266:  requires three integers per substring, whereas the POSIX interface provides
267:  only two. If the number of expected substrings is small, the wrapper
268:  function uses space on the stack, because this is faster than using
269:  malloc() for each call. The threshold above which the stack is no longer
270:  used is defined by POSIX_MALLOC_THRESHOLD. On systems that support it,
271:  "configure" can be used to override this default. */
272: #ifndef POSIX_MALLOC_THRESHOLD
273: #define POSIX_MALLOC_THRESHOLD 10
274: #endif
275:
276: /* Define to 1 if you have the ANSI C header files. */
277: #ifndef STDC_HEADERS
278: #define STDC_HEADERS 1
279: #endif
280:
281: /* Define to allow pcregrep to be linked with libbz2, so that it is able to
282:  handle .bz2 files. */
283: /* #undef SUPPORT_LIBBZ2 */
284:
285: /* Define to allow pcretest to be linked with libreadline. */

```

```
286: /* #undef SUPPORT_LIBREADLINE */
287:
288: /* Define to allow pcregrep to be linked with libz, so that it is able to
289:  handle .gz files. */
290: /* #undef SUPPORT_LIBZ */
291:
292: /* Define to enable support for Unicode properties */
293: /* #undef SUPPORT_UCP */
294:
295: /* Define to enable support for the UTF-8 Unicode encoding. */
296: /* #undef SUPPORT_UTF8 */
297:
298: /* Version number of package */
299: #ifndef VERSION
300: #define VERSION "7.8"
301: #endif
302:
303: /* Define to empty if `const' does not conform to ANSI C. */
304: /* #undef const */
305:
306: /* Define to `unsigned int' if <sys/types.h> does not define. */
307: /* #undef size_t */
```

File: sdm/VxWorks/libRegex/pcre_newline.c

```
1: /*****
2:  *   Perl-Compatible Regular Expressions   *
3: *****/
4:
5: /* PCRE is a library of functions to support regular expressions whose syntax
6: and semantics are as close as possible to those of the Perl 5 language.
7:
8:         Written by Philip Hazel
9:     Copyright (c) 1997-2008 University of Cambridge
10:
11: -----
12: Redistribution and use in source and binary forms, with or without
13: modification, are permitted provided that the following conditions are met:
14:
15:  * Redistributions of source code must retain the above copyright notice,
16:    this list of conditions and the following disclaimer.
17:
18:  * Redistributions in binary form must reproduce the above copyright
19:    notice, this list of conditions and the following disclaimer in the
20:    documentation and/or other materials provided with the distribution.
21:
22:  * Neither the name of the University of Cambridge nor the names of its
23:    contributors may be used to endorse or promote products derived from
24:    this software without specific prior written permission.
25:
26: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
27: "AS IS"
28: AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
29: THE
30: IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
31: PURPOSE
32: ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
33: BE
34: LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
35: CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
36: SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
37: INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
38: IN
39: CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
40: ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```

36: POSSIBILITY OF SUCH DAMAGE.
37: -----
38: */
39:
40:
41: /* This module contains internal functions for testing newlines when more than
42: one kind of newline is to be recognized. When a newline is found, its length is
43: returned. In principle, we could implement several newline "types", each
44: referring to a different set of newline characters. At present, PCRE supports
45: only NLTYPE_FIXED, which gets handled without these functions, NLTYPE_ANYCRLF,
46: and NLTYPE_ANY. The full list of Unicode newline characters is taken from
47: http://unicode.org/unicode/reports/tr18/. */
48:
49:
50: #ifdef HAVE_CONFIG_H
51: #include "config.h"
52: #endif
53:
54: #include "pcre_internal.h"
55:
56:
57:
58: /*****
59: *    Check for newline at given position    *
60: *****/
61:
62: /* It is guaranteed that the initial value of ptr is less than the end of the
63: string that is being processed.
64:
65: Arguments:
66: ptr      pointer to possible newline
67: type     the newline type
68: endptr   pointer to the end of the string
69: lenptr   where to return the length
70: utf8     TRUE if in utf8 mode
71:
72: Returns:  TRUE or FALSE
73: */
74:
75: BOOL
76: _pcre_is_newline(const uschar *ptr, int type, const uschar *endptr,

```

```

77: int *lenptr, BOOL utf8)
78: {
79: int c;
80: if (utf8) { GETCHAR(c, ptr); } else c = *ptr;
81:
82: if (type == NLTYPE_ANYCRLF) switch(c)
83: {
84: case 0x000a: *lenptr = 1; return TRUE;          /* LF */
85: case 0x000d: *lenptr = (ptr < endptr - 1 && ptr[1] == 0x0a)? 2 : 1;
86:             return TRUE;                        /* CR */
87: default: return FALSE;
88: }
89:
90: /* NLTYPE_ANY */
91:
92: else switch(c)
93: {
94: case 0x000a:                /* LF */
95: case 0x000b:                /* VT */
96: case 0x000c: *lenptr = 1; return TRUE;          /* FF */
97: case 0x000d: *lenptr = (ptr < endptr - 1 && ptr[1] == 0x0a)? 2 : 1;
98:             return TRUE;                        /* CR */
99: case 0x0085: *lenptr = utf8? 2 : 1; return TRUE; /* NEL */
100: case 0x2028:                /* LS */
101: case 0x2029: *lenptr = 3; return TRUE;          /* PS */
102: default: return FALSE;
103: }
104: }
105:
106:
107:
108: /*****
109: *   Check for newline at previous position   *
110: *****/
111:
112: /* It is guaranteed that the initial value of ptr is greater than the start of
113: the string that is being processed.
114:
115: Arguments:
116: ptr      pointer to possible newline
117: type     the newline type

```



```

118: startptr  pointer to the start of the string
119: lenptr    where to return the length
120: utf8      TRUE if in utf8 mode
121:
122: Returns:   TRUE or FALSE
123: */
124:
125: BOOL
126: _pcre_was_newline(const uschar *ptr, int type, const uschar *startptr,
127: int *lenptr, BOOL utf8)
128: {
129: int c;
130: ptr--;
131: #ifdef SUPPORT_UTF8
132: if (utf8)
133: {
134: BACKCHAR(ptr);
135: GETCHAR(c, ptr);
136: }
137: else c = *ptr;
138: #else /* no UTF-8 support */
139: c = *ptr;
140: #endif /* SUPPORT_UTF8 */
141:
142: if (type == NLTYPE_ANYCRLF) switch(c)
143: {
144: case 0x000a: *lenptr = (ptr > startptr && ptr[-1] == 0x0d)? 2 : 1;
145:             return TRUE;                /* LF */
146: case 0x000d: *lenptr = 1; return TRUE;    /* CR */
147: default: return FALSE;
148: }
149:
150: else switch(c)
151: {
152: case 0x000a: *lenptr = (ptr > startptr && ptr[-1] == 0x0d)? 2 : 1;
153:             return TRUE;                /* LF */
154: case 0x000b: /* VT */
155: case 0x000c: /* FF */
156: case 0x000d: *lenptr = 1; return TRUE;    /* CR */
157: case 0x0085: *lenptr = utf8? 2 : 1; return TRUE; /* NEL */
158: case 0x2028: /* LS */

```

```
159: case 0x2029: *lenptr = 3; return TRUE;      /* PS */
160: default: return FALSE;
161: }
162: }
163:
164: /* End of pcre_newline.c */
```

File: sdm/VxWorks/libRegex/pcre_ucl.c

```
1: #ifdef HAVE_CONFIG_H
2: #include "config.h"
3: #endif
4: #include "pcre_internal.h"
5:
6: /* Unicode character database. */
7: /* This file was autogenerated by the MultiStage2.py script. */
8: /* Total size: 52808 bytes, block size: 128. */
9: /* When recompiling tables with a new Unicode version,
10: please check types in the structure definition from pcre_internal.h:
11: typedef struct {
12: uschar property_0;
13: uschar property_1;
14: pcre_int32 property_2;
15: } ucd_record; */
16:
17:
18: const ucd_record _pcre_ucl_records[] = { /* 3656 bytes, record size 8 */
19: { 9, 0, 0, }, /* 0 */
20: { 9, 29, 0, }, /* 1 */
21: { 9, 21, 0, }, /* 2 */
22: { 9, 23, 0, }, /* 3 */
23: { 9, 22, 0, }, /* 4 */
24: { 9, 18, 0, }, /* 5 */
25: { 9, 25, 0, }, /* 6 */
26: { 9, 17, 0, }, /* 7 */
27: { 9, 13, 0, }, /* 8 */
28: { 33, 9, 32, }, /* 9 */
29: { 9, 24, 0, }, /* 10 */
30: { 9, 16, 0, }, /* 11 */
31: { 33, 5, -32, }, /* 12 */
32: { 9, 26, 0, }, /* 13 */
33: { 33, 5, 0, }, /* 14 */
34: { 9, 20, 0, }, /* 15 */
35: { 9, 1, 0, }, /* 16 */
36: { 9, 15, 0, }, /* 17 */
37: { 9, 5, 743, }, /* 18 */
38: { 9, 19, 0, }, /* 19 */
39: { 33, 5, 121, }, /* 20 */
```

40: { 33, 9, 1, }, /* 21 */
 41: { 33, 5, -1, }, /* 22 */
 42: { 33, 9, -199, }, /* 23 */
 43: { 33, 5, -232, }, /* 24 */
 44: { 33, 9, -121, }, /* 25 */
 45: { 33, 5, -300, }, /* 26 */
 46: { 33, 5, 195, }, /* 27 */
 47: { 33, 9, 210, }, /* 28 */
 48: { 33, 9, 206, }, /* 29 */
 49: { 33, 9, 205, }, /* 30 */
 50: { 33, 9, 79, }, /* 31 */
 51: { 33, 9, 202, }, /* 32 */
 52: { 33, 9, 203, }, /* 33 */
 53: { 33, 9, 207, }, /* 34 */
 54: { 33, 5, 97, }, /* 35 */
 55: { 33, 9, 211, }, /* 36 */
 56: { 33, 9, 209, }, /* 37 */
 57: { 33, 5, 163, }, /* 38 */
 58: { 33, 9, 213, }, /* 39 */
 59: { 33, 5, 130, }, /* 40 */
 60: { 33, 9, 214, }, /* 41 */
 61: { 33, 9, 218, }, /* 42 */
 62: { 33, 9, 217, }, /* 43 */
 63: { 33, 9, 219, }, /* 44 */
 64: { 33, 7, 0, }, /* 45 */
 65: { 33, 5, 56, }, /* 46 */
 66: { 33, 9, 2, }, /* 47 */
 67: { 33, 8, -1, }, /* 48 */
 68: { 33, 5, -2, }, /* 49 */
 69: { 33, 5, -79, }, /* 50 */
 70: { 33, 9, -97, }, /* 51 */
 71: { 33, 9, -56, }, /* 52 */
 72: { 33, 9, -130, }, /* 53 */
 73: { 33, 9, 10795, }, /* 54 */
 74: { 33, 9, -163, }, /* 55 */
 75: { 33, 9, 10792, }, /* 56 */
 76: { 33, 9, -195, }, /* 57 */
 77: { 33, 9, 69, }, /* 58 */
 78: { 33, 9, 71, }, /* 59 */
 79: { 33, 5, 10783, }, /* 60 */
 80: { 33, 5, 10780, }, /* 61 */

81: { 33, 5, -210, }, /* 62 */
 82: { 33, 5, -206, }, /* 63 */
 83: { 33, 5, -205, }, /* 64 */
 84: { 33, 5, -202, }, /* 65 */
 85: { 33, 5, -203, }, /* 66 */
 86: { 33, 5, -207, }, /* 67 */
 87: { 33, 5, -209, }, /* 68 */
 88: { 33, 5, -211, }, /* 69 */
 89: { 33, 5, 10743, }, /* 70 */
 90: { 33, 5, 10749, }, /* 71 */
 91: { 33, 5, -213, }, /* 72 */
 92: { 33, 5, -214, }, /* 73 */
 93: { 33, 5, 10727, }, /* 74 */
 94: { 33, 5, -218, }, /* 75 */
 95: { 33, 5, -69, }, /* 76 */
 96: { 33, 5, -217, }, /* 77 */
 97: { 33, 5, -71, }, /* 78 */
 98: { 33, 5, -219, }, /* 79 */
 99: { 33, 6, 0, }, /* 80 */
 100: { 9, 6, 0, }, /* 81 */
 101: { 27, 12, 0, }, /* 82 */
 102: { 27, 12, 84, }, /* 83 */
 103: { 19, 9, 1, }, /* 84 */
 104: { 19, 5, -1, }, /* 85 */
 105: { 19, 24, 0, }, /* 86 */
 106: { 9, 2, 0, }, /* 87 */
 107: { 19, 6, 0, }, /* 88 */
 108: { 19, 5, 130, }, /* 89 */
 109: { 19, 9, 38, }, /* 90 */
 110: { 19, 9, 37, }, /* 91 */
 111: { 19, 9, 64, }, /* 92 */
 112: { 19, 9, 63, }, /* 93 */
 113: { 19, 5, 0, }, /* 94 */
 114: { 19, 9, 32, }, /* 95 */
 115: { 19, 5, -38, }, /* 96 */
 116: { 19, 5, -37, }, /* 97 */
 117: { 19, 5, -32, }, /* 98 */
 118: { 19, 5, -31, }, /* 99 */
 119: { 19, 5, -64, }, /* 100 */
 120: { 19, 5, -63, }, /* 101 */
 121: { 19, 9, 8, }, /* 102 */

122: { 19, 5, -62, }, /* 103 */
 123: { 19, 5, -57, }, /* 104 */
 124: { 19, 9, 0, }, /* 105 */
 125: { 19, 5, -47, }, /* 106 */
 126: { 19, 5, -54, }, /* 107 */
 127: { 19, 5, -8, }, /* 108 */
 128: { 10, 9, 1, }, /* 109 */
 129: { 10, 5, -1, }, /* 110 */
 130: { 19, 5, -86, }, /* 111 */
 131: { 19, 5, -80, }, /* 112 */
 132: { 19, 5, 7, }, /* 113 */
 133: { 19, 9, -60, }, /* 114 */
 134: { 19, 5, -96, }, /* 115 */
 135: { 19, 25, 0, }, /* 116 */
 136: { 19, 9, -7, }, /* 117 */
 137: { 19, 9, -130, }, /* 118 */
 138: { 12, 9, 80, }, /* 119 */
 139: { 12, 9, 32, }, /* 120 */
 140: { 12, 5, -32, }, /* 121 */
 141: { 12, 5, -80, }, /* 122 */
 142: { 12, 9, 1, }, /* 123 */
 143: { 12, 5, -1, }, /* 124 */
 144: { 12, 26, 0, }, /* 125 */
 145: { 12, 12, 0, }, /* 126 */
 146: { 12, 11, 0, }, /* 127 */
 147: { 12, 9, 15, }, /* 128 */
 148: { 12, 5, -15, }, /* 129 */
 149: { 1, 9, 48, }, /* 130 */
 150: { 1, 6, 0, }, /* 131 */
 151: { 1, 21, 0, }, /* 132 */
 152: { 1, 5, -48, }, /* 133 */
 153: { 1, 5, 0, }, /* 134 */
 154: { 1, 17, 0, }, /* 135 */
 155: { 25, 12, 0, }, /* 136 */
 156: { 25, 17, 0, }, /* 137 */
 157: { 25, 21, 0, }, /* 138 */
 158: { 25, 7, 0, }, /* 139 */
 159: { 0, 25, 0, }, /* 140 */
 160: { 0, 21, 0, }, /* 141 */
 161: { 0, 23, 0, }, /* 142 */
 162: { 0, 26, 0, }, /* 143 */

163: { 0, 12, 0, }, /* 144 */
 164: { 0, 7, 0, }, /* 145 */
 165: { 0, 11, 0, }, /* 146 */
 166: { 0, 6, 0, }, /* 147 */
 167: { 0, 13, 0, }, /* 148 */
 168: { 49, 21, 0, }, /* 149 */
 169: { 49, 1, 0, }, /* 150 */
 170: { 49, 7, 0, }, /* 151 */
 171: { 49, 12, 0, }, /* 152 */
 172: { 55, 7, 0, }, /* 153 */
 173: { 55, 12, 0, }, /* 154 */
 174: { 63, 13, 0, }, /* 155 */
 175: { 63, 7, 0, }, /* 156 */
 176: { 63, 12, 0, }, /* 157 */
 177: { 63, 6, 0, }, /* 158 */
 178: { 63, 26, 0, }, /* 159 */
 179: { 63, 21, 0, }, /* 160 */
 180: { 14, 12, 0, }, /* 161 */
 181: { 14, 10, 0, }, /* 162 */
 182: { 14, 7, 0, }, /* 163 */
 183: { 14, 13, 0, }, /* 164 */
 184: { 14, 6, 0, }, /* 165 */
 185: { 2, 12, 0, }, /* 166 */
 186: { 2, 10, 0, }, /* 167 */
 187: { 2, 7, 0, }, /* 168 */
 188: { 2, 13, 0, }, /* 169 */
 189: { 2, 23, 0, }, /* 170 */
 190: { 2, 15, 0, }, /* 171 */
 191: { 2, 26, 0, }, /* 172 */
 192: { 21, 12, 0, }, /* 173 */
 193: { 21, 10, 0, }, /* 174 */
 194: { 21, 7, 0, }, /* 175 */
 195: { 21, 13, 0, }, /* 176 */
 196: { 20, 12, 0, }, /* 177 */
 197: { 20, 10, 0, }, /* 178 */
 198: { 20, 7, 0, }, /* 179 */
 199: { 20, 13, 0, }, /* 180 */
 200: { 20, 23, 0, }, /* 181 */
 201: { 43, 12, 0, }, /* 182 */
 202: { 43, 10, 0, }, /* 183 */
 203: { 43, 7, 0, }, /* 184 */

204: { 43, 13, 0, }, /* 185 */
 205: { 43, 26, 0, }, /* 186 */
 206: { 53, 12, 0, }, /* 187 */
 207: { 53, 7, 0, }, /* 188 */
 208: { 53, 10, 0, }, /* 189 */
 209: { 53, 13, 0, }, /* 190 */
 210: { 53, 15, 0, }, /* 191 */
 211: { 53, 26, 0, }, /* 192 */
 212: { 53, 23, 0, }, /* 193 */
 213: { 54, 10, 0, }, /* 194 */
 214: { 54, 7, 0, }, /* 195 */
 215: { 54, 12, 0, }, /* 196 */
 216: { 54, 13, 0, }, /* 197 */
 217: { 54, 15, 0, }, /* 198 */
 218: { 54, 26, 0, }, /* 199 */
 219: { 28, 10, 0, }, /* 200 */
 220: { 28, 7, 0, }, /* 201 */
 221: { 28, 12, 0, }, /* 202 */
 222: { 28, 13, 0, }, /* 203 */
 223: { 36, 10, 0, }, /* 204 */
 224: { 36, 7, 0, }, /* 205 */
 225: { 36, 12, 0, }, /* 206 */
 226: { 36, 13, 0, }, /* 207 */
 227: { 36, 15, 0, }, /* 208 */
 228: { 36, 26, 0, }, /* 209 */
 229: { 47, 10, 0, }, /* 210 */
 230: { 47, 7, 0, }, /* 211 */
 231: { 47, 12, 0, }, /* 212 */
 232: { 47, 21, 0, }, /* 213 */
 233: { 56, 7, 0, }, /* 214 */
 234: { 56, 12, 0, }, /* 215 */
 235: { 56, 6, 0, }, /* 216 */
 236: { 56, 21, 0, }, /* 217 */
 237: { 56, 13, 0, }, /* 218 */
 238: { 32, 7, 0, }, /* 219 */
 239: { 32, 12, 0, }, /* 220 */
 240: { 32, 6, 0, }, /* 221 */
 241: { 32, 13, 0, }, /* 222 */
 242: { 57, 7, 0, }, /* 223 */
 243: { 57, 26, 0, }, /* 224 */
 244: { 57, 21, 0, }, /* 225 */

245: { 57, 12, 0, }, /* 226 */
 246: { 57, 13, 0, }, /* 227 */
 247: { 57, 15, 0, }, /* 228 */
 248: { 57, 22, 0, }, /* 229 */
 249: { 57, 18, 0, }, /* 230 */
 250: { 57, 10, 0, }, /* 231 */
 251: { 38, 7, 0, }, /* 232 */
 252: { 38, 10, 0, }, /* 233 */
 253: { 38, 12, 0, }, /* 234 */
 254: { 38, 13, 0, }, /* 235 */
 255: { 38, 21, 0, }, /* 236 */
 256: { 38, 26, 0, }, /* 237 */
 257: { 16, 9, 7264, }, /* 238 */
 258: { 16, 7, 0, }, /* 239 */
 259: { 16, 6, 0, }, /* 240 */
 260: { 23, 7, 0, }, /* 241 */
 261: { 15, 7, 0, }, /* 242 */
 262: { 15, 12, 0, }, /* 243 */
 263: { 15, 26, 0, }, /* 244 */
 264: { 15, 21, 0, }, /* 245 */
 265: { 15, 15, 0, }, /* 246 */
 266: { 8, 7, 0, }, /* 247 */
 267: { 7, 7, 0, }, /* 248 */
 268: { 7, 21, 0, }, /* 249 */
 269: { 40, 29, 0, }, /* 250 */
 270: { 40, 7, 0, }, /* 251 */
 271: { 40, 22, 0, }, /* 252 */
 272: { 40, 18, 0, }, /* 253 */
 273: { 45, 7, 0, }, /* 254 */
 274: { 45, 14, 0, }, /* 255 */
 275: { 50, 7, 0, }, /* 256 */
 276: { 50, 12, 0, }, /* 257 */
 277: { 24, 7, 0, }, /* 258 */
 278: { 24, 12, 0, }, /* 259 */
 279: { 6, 7, 0, }, /* 260 */
 280: { 6, 12, 0, }, /* 261 */
 281: { 51, 7, 0, }, /* 262 */
 282: { 51, 12, 0, }, /* 263 */
 283: { 31, 7, 0, }, /* 264 */
 284: { 31, 1, 0, }, /* 265 */
 285: { 31, 10, 0, }, /* 266 */

286: { 31, 12, 0, }, /* 267 */
 287: { 31, 21, 0, }, /* 268 */
 288: { 31, 6, 0, }, /* 269 */
 289: { 31, 23, 0, }, /* 270 */
 290: { 31, 13, 0, }, /* 271 */
 291: { 31, 15, 0, }, /* 272 */
 292: { 37, 21, 0, }, /* 273 */
 293: { 37, 17, 0, }, /* 274 */
 294: { 37, 12, 0, }, /* 275 */
 295: { 37, 29, 0, }, /* 276 */
 296: { 37, 13, 0, }, /* 277 */
 297: { 37, 7, 0, }, /* 278 */
 298: { 37, 6, 0, }, /* 279 */
 299: { 34, 7, 0, }, /* 280 */
 300: { 34, 12, 0, }, /* 281 */
 301: { 34, 10, 0, }, /* 282 */
 302: { 34, 26, 0, }, /* 283 */
 303: { 34, 21, 0, }, /* 284 */
 304: { 34, 13, 0, }, /* 285 */
 305: { 52, 7, 0, }, /* 286 */
 306: { 39, 7, 0, }, /* 287 */
 307: { 39, 10, 0, }, /* 288 */
 308: { 39, 13, 0, }, /* 289 */
 309: { 39, 21, 0, }, /* 290 */
 310: { 31, 26, 0, }, /* 291 */
 311: { 5, 7, 0, }, /* 292 */
 312: { 5, 12, 0, }, /* 293 */
 313: { 5, 10, 0, }, /* 294 */
 314: { 5, 21, 0, }, /* 295 */
 315: { 61, 12, 0, }, /* 296 */
 316: { 61, 10, 0, }, /* 297 */
 317: { 61, 7, 0, }, /* 298 */
 318: { 61, 13, 0, }, /* 299 */
 319: { 61, 21, 0, }, /* 300 */
 320: { 61, 26, 0, }, /* 301 */
 321: { 75, 12, 0, }, /* 302 */
 322: { 75, 10, 0, }, /* 303 */
 323: { 75, 7, 0, }, /* 304 */
 324: { 75, 13, 0, }, /* 305 */
 325: { 69, 7, 0, }, /* 306 */
 326: { 69, 10, 0, }, /* 307 */

327: { 69, 12, 0, }, /* 308 */
 328: { 69, 21, 0, }, /* 309 */
 329: { 69, 13, 0, }, /* 310 */
 330: { 72, 13, 0, }, /* 311 */
 331: { 72, 7, 0, }, /* 312 */
 332: { 72, 6, 0, }, /* 313 */
 333: { 72, 21, 0, }, /* 314 */
 334: { 12, 5, 0, }, /* 315 */
 335: { 12, 6, 0, }, /* 316 */
 336: { 33, 5, 35332, }, /* 317 */
 337: { 33, 5, 3814, }, /* 318 */
 338: { 33, 5, -59, }, /* 319 */
 339: { 33, 9, -7615, }, /* 320 */
 340: { 19, 5, 8, }, /* 321 */
 341: { 19, 9, -8, }, /* 322 */
 342: { 19, 5, 74, }, /* 323 */
 343: { 19, 5, 86, }, /* 324 */
 344: { 19, 5, 100, }, /* 325 */
 345: { 19, 5, 128, }, /* 326 */
 346: { 19, 5, 112, }, /* 327 */
 347: { 19, 5, 126, }, /* 328 */
 348: { 19, 8, -8, }, /* 329 */
 349: { 19, 5, 9, }, /* 330 */
 350: { 19, 9, -74, }, /* 331 */
 351: { 19, 8, -9, }, /* 332 */
 352: { 19, 5, -7205, }, /* 333 */
 353: { 19, 9, -86, }, /* 334 */
 354: { 19, 9, -100, }, /* 335 */
 355: { 19, 9, -112, }, /* 336 */
 356: { 19, 9, -128, }, /* 337 */
 357: { 19, 9, -126, }, /* 338 */
 358: { 27, 1, 0, }, /* 339 */
 359: { 9, 27, 0, }, /* 340 */
 360: { 9, 28, 0, }, /* 341 */
 361: { 27, 11, 0, }, /* 342 */
 362: { 9, 9, 0, }, /* 343 */
 363: { 9, 5, 0, }, /* 344 */
 364: { 19, 9, -7517, }, /* 345 */
 365: { 33, 9, -8383, }, /* 346 */
 366: { 33, 9, -8262, }, /* 347 */
 367: { 33, 9, 28, }, /* 348 */

368: { 9, 7, 0, }, /* 349 */
 369: { 33, 5, -28, }, /* 350 */
 370: { 33, 14, 16, }, /* 351 */
 371: { 33, 14, -16, }, /* 352 */
 372: { 33, 14, 0, }, /* 353 */
 373: { 9, 26, 26, }, /* 354 */
 374: { 9, 26, -26, }, /* 355 */
 375: { 4, 26, 0, }, /* 356 */
 376: { 17, 9, 48, }, /* 357 */
 377: { 17, 5, -48, }, /* 358 */
 378: { 33, 9, -10743, }, /* 359 */
 379: { 33, 9, -3814, }, /* 360 */
 380: { 33, 9, -10727, }, /* 361 */
 381: { 33, 5, -10795, }, /* 362 */
 382: { 33, 5, -10792, }, /* 363 */
 383: { 33, 9, -10780, }, /* 364 */
 384: { 33, 9, -10749, }, /* 365 */
 385: { 33, 9, -10783, }, /* 366 */
 386: { 10, 5, 0, }, /* 367 */
 387: { 10, 26, 0, }, /* 368 */
 388: { 10, 21, 0, }, /* 369 */
 389: { 10, 15, 0, }, /* 370 */
 390: { 16, 5, -7264, }, /* 371 */
 391: { 58, 7, 0, }, /* 372 */
 392: { 58, 6, 0, }, /* 373 */
 393: { 22, 26, 0, }, /* 374 */
 394: { 22, 6, 0, }, /* 375 */
 395: { 22, 14, 0, }, /* 376 */
 396: { 26, 7, 0, }, /* 377 */
 397: { 26, 6, 0, }, /* 378 */
 398: { 29, 7, 0, }, /* 379 */
 399: { 29, 6, 0, }, /* 380 */
 400: { 3, 7, 0, }, /* 381 */
 401: { 23, 26, 0, }, /* 382 */
 402: { 29, 26, 0, }, /* 383 */
 403: { 22, 7, 0, }, /* 384 */
 404: { 60, 7, 0, }, /* 385 */
 405: { 60, 6, 0, }, /* 386 */
 406: { 60, 26, 0, }, /* 387 */
 407: { 76, 7, 0, }, /* 388 */
 408: { 76, 6, 0, }, /* 389 */

409: { 76, 21, 0, }, /* 390 */
 410: { 76, 13, 0, }, /* 391 */
 411: { 12, 7, 0, }, /* 392 */
 412: { 12, 21, 0, }, /* 393 */
 413: { 33, 9, -35332, }, /* 394 */
 414: { 48, 7, 0, }, /* 395 */
 415: { 48, 12, 0, }, /* 396 */
 416: { 48, 10, 0, }, /* 397 */
 417: { 48, 26, 0, }, /* 398 */
 418: { 64, 7, 0, }, /* 399 */
 419: { 64, 21, 0, }, /* 400 */
 420: { 74, 10, 0, }, /* 401 */
 421: { 74, 7, 0, }, /* 402 */
 422: { 74, 12, 0, }, /* 403 */
 423: { 74, 21, 0, }, /* 404 */
 424: { 74, 13, 0, }, /* 405 */
 425: { 68, 13, 0, }, /* 406 */
 426: { 68, 7, 0, }, /* 407 */
 427: { 68, 12, 0, }, /* 408 */
 428: { 68, 21, 0, }, /* 409 */
 429: { 73, 7, 0, }, /* 410 */
 430: { 73, 12, 0, }, /* 411 */
 431: { 73, 10, 0, }, /* 412 */
 432: { 73, 21, 0, }, /* 413 */
 433: { 67, 7, 0, }, /* 414 */
 434: { 67, 12, 0, }, /* 415 */
 435: { 67, 10, 0, }, /* 416 */
 436: { 67, 13, 0, }, /* 417 */
 437: { 67, 21, 0, }, /* 418 */
 438: { 9, 4, 0, }, /* 419 */
 439: { 9, 3, 0, }, /* 420 */
 440: { 25, 25, 0, }, /* 421 */
 441: { 35, 7, 0, }, /* 422 */
 442: { 19, 14, 0, }, /* 423 */
 443: { 19, 15, 0, }, /* 424 */
 444: { 19, 26, 0, }, /* 425 */
 445: { 70, 7, 0, }, /* 426 */
 446: { 66, 7, 0, }, /* 427 */
 447: { 41, 7, 0, }, /* 428 */
 448: { 41, 15, 0, }, /* 429 */
 449: { 18, 7, 0, }, /* 430 */

```

450: { 18, 14, 0, }, /* 431 */
451: { 59, 7, 0, }, /* 432 */
452: { 59, 21, 0, }, /* 433 */
453: { 42, 7, 0, }, /* 434 */
454: { 42, 21, 0, }, /* 435 */
455: { 42, 14, 0, }, /* 436 */
456: { 13, 9, 40, }, /* 437 */
457: { 13, 5, -40, }, /* 438 */
458: { 46, 7, 0, }, /* 439 */
459: { 44, 7, 0, }, /* 440 */
460: { 44, 13, 0, }, /* 441 */
461: { 11, 7, 0, }, /* 442 */
462: { 65, 7, 0, }, /* 443 */
463: { 65, 15, 0, }, /* 444 */
464: { 65, 21, 0, }, /* 445 */
465: { 71, 7, 0, }, /* 446 */
466: { 71, 21, 0, }, /* 447 */
467: { 30, 7, 0, }, /* 448 */
468: { 30, 12, 0, }, /* 449 */
469: { 30, 15, 0, }, /* 450 */
470: { 30, 21, 0, }, /* 451 */
471: { 62, 7, 0, }, /* 452 */
472: { 62, 14, 0, }, /* 453 */
473: { 62, 21, 0, }, /* 454 */
474: { 9, 10, 0, }, /* 455 */
475: { 19, 12, 0, }, /* 456 */
476: };
477:
478: const uschar _pcre_ucl_stage1[] = { /* 8704 bytes */
479: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, /* U+0000 */
480: 16, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, /* U+0800 */
481: 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 40, 40, 41, 42, 43, 44, /* U+1000 */
482: 45, 46, 47, 48, 49, 16, 50, 51, 52, 16, 53, 54, 55, 56, 57, 58, /* U+1800 */
483: 59, 60, 61, 62, 63, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, /* U+2000 */
484: 74, 74, 63, 75, 63, 63, 76, 16, 77, 78, 79, 80, 81, 82, 83, 84, /* U+2800 */
485: 85, 86, 87, 88, 89, 90, 91, 68, 92, 92, 92, 92, 92, 92, 92, 92, /* U+3000 */
486: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+3800 */
487: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+4000 */
488: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 93, 92, 92, 92, 92, /* U+4800 */
489: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+5000 */
490: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+5800 */

```

491: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+6000 */
492: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+6800 */
493: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+7000 */
494: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+7800 */
495: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+8000 */
496: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+8800 */
497: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+9000 */
498: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 94, /* U+9800 */
499: 95, 96, 96, 96, 96, 96, 96, 96, 96, 97, 98, 98, 99, 100, 101, 102, /* U+A000 */
500: 103, 104, 105, 16, 106, 16, 16, 16, 107, 107, 107, 107, 107, 107, 107, 107, /* U+A800 */
501: 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, /* U+B000 */
502: 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, /* U+B800 */
503: 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, /* U+C000 */
504: 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, /* U+C800 */
505: 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 107, 108, /* U+D000 */
506: 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, 109, /* U+D800 */
507: 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, /* U+E000 */
508: 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, /* U+E800 */
509: 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, /* U+F000 */
510: 110, 110, 92, 92, 111, 112, 113, 114, 115, 115, 116, 117, 118, 119, 120, 121, /* U+F800 */
511: 122, 123, 124, 125, 16, 126, 127, 128, 129, 130, 16, 16, 16, 16, 16, 16, /* U+10000 */
512: 131, 16, 132, 16, 133, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+10800 */
513: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+11000 */
514: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+11800 */
515: 134, 134, 134, 134, 134, 134, 135, 16, 136, 16, 16, 16, 16, 16, 16, 16, /* U+12000 */
516: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+12800 */
517: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+13000 */
518: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+13800 */
519: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+14000 */
520: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+14800 */
521: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+15000 */
522: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+15800 */
523: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+16000 */
524: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+16800 */
525: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+17000 */
526: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+17800 */
527: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+18000 */
528: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+18800 */
529: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+19000 */
530: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+19800 */
531: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1A000 */

532: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1A800 */
533: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1B000 */
534: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1B800 */
535: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1C000 */
536: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1C800 */
537: 68,137,138,139,140, 16,141, 16,142,143,144,145,146,147,148,149, /* U+1D000 */
538: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1D800 */
539: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1E000 */
540: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1E800 */
541: 150,151, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1F000 */
542: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+1F800 */
543: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+20000 */
544: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+20800 */
545: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+21000 */
546: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+21800 */
547: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+22000 */
548: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+22800 */
549: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+23000 */
550: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+23800 */
551: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+24000 */
552: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+24800 */
553: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+25000 */
554: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+25800 */
555: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+26000 */
556: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+26800 */
557: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+27000 */
558: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+27800 */
559: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+28000 */
560: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+28800 */
561: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+29000 */
562: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, /* U+29800 */
563: 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92, 92,152, 16, 16, /* U+2A000 */
564: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2A800 */
565: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2B000 */
566: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2B800 */
567: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2C000 */
568: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2C800 */
569: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2D000 */
570: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2D800 */
571: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2E000 */
572: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, /* U+2E800 */

3215
Approved for public release; distribution is unlimited

[illegible]

3217
Approved for public release; distribution is unlimited

3218
Approved for public release; distribution is unlimited

[illegible]

3220
Approved for public release; distribution is unlimited

[illegible]

[illegible]

3223
Approved for public release; distribution is unlimited

3224
Approved for public release; distribution is unlimited

983: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+FC000 */
984: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+FC800 */
985: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+FD000 */
986: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+FD800 */
987: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+FE000 */
988: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+FE800 */
989: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+FF000 */
990: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,157, /* U+FF800 */
991: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+100000 */
992: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+100800 */
993: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+101000 */
994: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+101800 */
995: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+102000 */
996: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+102800 */
997: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+103000 */
998: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+103800 */
999: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+104000 */
1000: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+104800 */
1001: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+105000 */
1002: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+105800 */
1003: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+106000 */
1004: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+106800 */
1005: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+107000 */
1006: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+107800 */
1007: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+108000 */
1008: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+108800 */
1009: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+109000 */
1010: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+109800 */
1011: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10A000 */
1012: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10A800 */
1013: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10B000 */
1014: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10B800 */
1015: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10C000 */
1016: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10C800 */
1017: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10D000 */
1018: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10D800 */
1019: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10E000 */
1020: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10E800 */
1021: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,110, /* U+10F000 */
1022: 110,110,110,110,110,110,110,110,110,110,110,110,110,110,157, /* U+10F800 */
1023: };

```

1024:
1025: const pcre_uint16 _pcre_ucd_stage2[] = { /* 40448 bytes, block = 128 */
1026: /* block 0 */
1027: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1028: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1029: 1, 2, 2, 2, 3, 2, 2, 2, 4, 5, 2, 6, 2, 7, 2, 2,
1030: 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 2, 2, 6, 6, 6, 2,
1031: 2, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
1032: 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 4, 2, 5, 10, 11,
1033: 10, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
1034: 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 4, 6, 5, 6, 0,
1035:
1036: /* block 1 */
1037: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1038: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1039: 1, 2, 3, 3, 3, 3, 13, 13, 10, 13, 14, 15, 6, 16, 13, 10,
1040: 13, 6, 17, 17, 10, 18, 13, 2, 10, 17, 14, 19, 17, 17, 17, 2,
1041: 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
1042: 9, 9, 9, 9, 9, 9, 9, 6, 9, 9, 9, 9, 9, 9, 9, 14,
1043: 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
1044: 12, 12, 12, 12, 12, 12, 12, 6, 12, 12, 12, 12, 12, 12, 12, 20,
1045:
1046: /* block 2 */
1047: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
1048: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
1049: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
1050: 23, 24, 21, 22, 21, 22, 21, 22, 21, 22, 14, 21, 22, 21, 22, 21, 22, 21,
1051: 22, 21, 22, 21, 22, 21, 22, 21, 22, 14, 21, 22, 21, 22, 21, 22,
1052: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
1053: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
1054: 21, 22, 21, 22, 21, 22, 21, 22, 25, 21, 22, 21, 22, 21, 22, 26,
1055:
1056: /* block 3 */
1057: 27, 28, 21, 22, 21, 22, 29, 21, 22, 30, 30, 21, 22, 14, 31, 32,
1058: 33, 21, 22, 30, 34, 35, 36, 37, 21, 22, 38, 14, 36, 39, 40, 41,
1059: 21, 22, 21, 22, 21, 22, 42, 21, 22, 42, 14, 14, 21, 22, 42, 21,
1060: 22, 43, 43, 21, 22, 21, 22, 44, 21, 22, 14, 45, 21, 22, 14, 46,
1061: 45, 45, 45, 45, 47, 48, 49, 47, 48, 49, 47, 48, 49, 21, 22, 21,
1062: 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 50, 21, 22,
1063: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
1064: 14, 47, 48, 49, 21, 22, 51, 52, 21, 22, 21, 22, 21, 22, 21, 22,

```

1065:

1066: /* block 4 */

1067: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,

1068: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,

1069: 53, 14, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,

1070: 21, 22, 21, 22, 14, 14, 14, 14, 14, 14, 54, 21, 22, 55, 56, 14,

1071: 14, 21, 22, 57, 58, 59, 21, 22, 21, 22, 21, 22, 21, 22,

1072: 60, 61, 14, 62, 63, 14, 64, 64, 14, 65, 14, 66, 14, 14, 14,

1073: 64, 14, 14, 67, 14, 14, 14, 14, 68, 69, 14, 70, 14, 14, 14, 69,

1074: 14, 71, 72, 14, 14, 73, 14, 14, 14, 14, 14, 14, 14, 74, 14, 14,

1075:

1076: /* block 5 */

1077: 75, 14, 14, 75, 14, 14, 14, 14, 75, 76, 77, 77, 78, 14, 14, 14,

1078: 14, 14, 79, 14, 45, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,

1079: 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,

1080: 80, 80, 80, 80, 80, 80, 80, 80, 80, 81, 81, 81, 81, 81, 81, 81,

1081: 81, 81, 10, 10, 10, 10, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81,

1082: 81, 81, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,

1083: 80, 80, 80, 80, 80, 10, 10, 10, 10, 10, 10, 10, 81, 10, 81, 10,

1084: 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,

1085:

1086: /* block 6 */

1087: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,

1088: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,

1089: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,

1090: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,

1091: 82, 82, 82, 82, 82, 83, 82, 82, 82, 82, 82, 82, 82, 82, 82,

1092: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,

1093: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,

1094: 84, 85, 84, 85, 81, 86, 84, 85, 87, 87, 88, 89, 89, 89, 2, 87,

1095:

1096: /* block 7 */

1097: 87, 87, 87, 87, 86, 10, 90, 2, 91, 91, 91, 87, 92, 87, 93, 93,

1098: 94, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95,

1099: 95, 95, 87, 95, 95, 95, 95, 95, 95, 95, 95, 96, 97, 97, 97,

1100: 94, 98, 98, 98, 98, 98, 98, 98, 98, 98, 98, 98, 98, 98, 98,

1101: 98, 98, 99, 98, 98, 98, 98, 98, 98, 98, 98, 98, 100, 101, 101, 102,

1102: 103, 104, 105, 105, 105, 106, 107, 108, 84, 85, 84, 85, 84, 85, 84, 85,

1103: 84, 85, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 110,

1104: 111, 112, 113, 94, 114, 115, 116, 84, 85, 117, 84, 85, 94, 118, 118, 118,

1105:

1106: /* block 8 */
1107: 119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,
1108: 120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,
1109: 120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,
1110: 121,121,121,121,121,121,121,121,121,121,121,121,121,121,121,
1111: 121,121,121,121,121,121,121,121,121,121,121,121,121,121,121,
1112: 122,122,122,122,122,122,122,122,122,122,122,122,122,122,122,
1113: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1114: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1115:
1116: /* block 9 */
1117: 123,124,125,126,126,126,126,126,127,127,123,124,123,124,123,124,
1118: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1119: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1120: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1121: 128,123,124,123,124,123,124,123,124,123,124,123,124,123,124,
1122: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1123: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1124: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1125:
1126: /* block 10 */
1127: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1128: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,
1129: 123,124,123,124, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1130: 87,130,130,130,130,130,130,130,130,130,130,130,130,130,130,
1131: 130,130,130,130,130,130,130,130,130,130,130,130,130,130,130,
1132: 130,130,130,130,130,130,130, 87, 87,131,132,132,132,132,132,132,
1133: 87,133,133,133,133,133,133,133,133,133,133,133,133,133,133,
1134: 133,133,133,133,133,133,133,133,133,133,133,133,133,133,133,
1135:
1136: /* block 11 */
1137: 133,133,133,133,133,133,133,134, 87, 2,135, 87, 87, 87, 87, 87,
1138: 87,136,136,136,136,136,136,136,136,136,136,136,136,136,136,
1139: 136,136,136,136,136,136,136,136,136,136,136,136,136,136,136,
1140: 136,136,136,136,136,136,136,136,136,136,136,136,136,137,136,
1141: 138,136,136,138,136,136,138,136, 87, 87, 87, 87, 87, 87, 87, 87,
1142: 139,139,139,139,139,139,139,139,139,139,139,139,139,139,139,
1143: 139,139,139,139,139,139,139,139,139,139, 87, 87, 87, 87, 87,
1144: 139,139,139,138,138, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1145:
1146: /* block 12 */

1147: 16, 16, 16, 16, 87, 87,140,140,140,141,141,142, 2,141,143,143,
1148: 144,144,144,144,144,144,144,144,144,144, 2, 87, 87,141, 2,
1149: 87,145,145,145,145,145,145,145,145,145,145,145,145,145,
1150: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1151: 81,145,145,145,145,145,145,145,145,145, 82, 82, 82, 82, 82,
1152: 82, 82, 82, 82, 82, 82,144,144,144,144,144,144,144,144, 87,
1153: 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,141,141,141,141,145,145,
1154: 82,145,145,145,145,145,145,145,145,145,145,145,145,145,
1155:
1156: /* block 13 */
1157: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1158: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1159: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1160: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1161: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1162: 145,145,145,145,141,145,144,144,144,144,144,144,144, 16,146,144,
1163: 144,144,144,144,144,147,147,144,144,143,144,144,144,144,145,145,
1164: 148,148,148,148,148,148,148,148,148,148,145,145,145,143,143,145,
1165:
1166: /* block 14 */
1167: 149,149,149,149,149,149,149,149,149,149,149,149,149, 87,150,
1168: 151,152,151,151,151,151,151,151,151,151,151,151,151,151,151,
1169: 151,151,151,151,151,151,151,151,151,151,151,151,151,151,151,
1170: 152,152,152,152,152,152,152,152,152,152,152,152,152,152,152,
1171: 152,152,152,152,152,152,152,152,152,152, 87, 87,151,151,151,
1172: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1173: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1174: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
1175:
1176: /* block 15 */
1177: 153,153,153,153,153,153,153,153,153,153,153,153,153,153,153,
1178: 153,153,153,153,153,153,153,153,153,153,153,153,153,153,153,
1179: 153,153,153,153,153,153,154,154,154,154,154,154,154,154,154,
1180: 154,153, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1181: 155,155,155,155,155,155,155,155,155,155,156,156,156,156,156,
1182: 156,156,156,156,156,156,156,156,156,156,156,156,156,156,156,
1183: 156,156,156,156,156,156,156,156,156,156,157,157,157,157,157,
1184: 157,157,157,157,158,158,159,160,160,160,160,158, 87, 87, 87, 87, 87,
1185:
1186: /* block 16 */
1187: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

1188: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1189: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1190: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1191: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1192: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1193: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1194: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1195:
1196: /* block 17 */
1197: 87,161,161,162,163,163,163,163,163,163,163,163,163,163,163,
1198: 163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,
1199: 163,163,163,163,163,163,163,163,163,163,163,163,163,163,163,
1200: 163,163,163,163,163,163,163,163,163,163, 87, 87,161,163,162,162,
1201: 162,161,161,161,161,161,161,161,161,162,162,162,162,161, 87, 87,
1202: 163, 82, 82,161,161, 87, 87, 87,163,163,163,163,163,163,163,163,
1203: 163,163,161,161, 2, 2,164,164,164,164,164,164,164,164,164,164,
1204: 2,165,163, 87, 87, 87, 87, 87, 87, 87, 87,163,163,163,163,163,
1205:
1206: /* block 18 */
1207: 87,166,167,167, 87,168,168,168,168,168,168,168,168, 87, 87,168,
1208: 168, 87, 87,168,168,168,168,168,168,168,168,168,168,168,168,
1209: 168,168,168,168,168,168,168,168,168, 87,168,168,168,168,168,168,
1210: 168, 87,168, 87, 87, 87,168,168,168,168, 87, 87,166,168,167,167,
1211: 167,166,166,166,166, 87, 87,167,167, 87, 87,167,167,166,168, 87,
1212: 87, 87, 87, 87, 87, 87, 87,167, 87, 87, 87, 87,168,168, 87,168,
1213: 168,168,166,166, 87, 87,169,169,169,169,169,169,169,169,169,169,
1214: 168,168,170,170,171,171,171,171,171,171,172, 87, 87, 87, 87, 87,
1215:
1216: /* block 19 */
1217: 87,173,173,174, 87,175,175,175,175,175,175,175, 87, 87, 87, 87,175,
1218: 175, 87, 87,175,175,175,175,175,175,175,175,175,175,175,175,
1219: 175,175,175,175,175,175,175,175,175,175, 87,175,175,175,175,175,175,
1220: 175, 87,175,175, 87,175,175, 87,175,175, 87, 87,173, 87,174,174,
1221: 174,173,173, 87, 87, 87, 87,173,173, 87, 87,173,173,173, 87, 87,
1222: 87,173, 87, 87, 87, 87, 87, 87, 87,175,175,175,175, 87,175, 87,
1223: 87, 87, 87, 87, 87, 87,176,176,176,176,176,176,176,176,176,176,
1224: 173,173,175,175,175,173, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1225:
1226: /* block 20 */
1227: 87,177,177,178, 87,179,179,179,179,179,179,179,179,179, 87,179,
1228: 179,179, 87,179,179,179,179,179,179,179,179,179,179,179,179,

1229: 179,179,179,179,179,179,179,179,179, 87,179,179,179,179,179,179,
1230: 179, 87,179,179, 87,179,179,179,179,179, 87, 87,177,179,178,178,
1231: 178,177,177,177,177,177, 87,177,177,178, 87,178,178,177, 87, 87,
1232: 179, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1233: 179,179,177,177, 87, 87,180,180,180,180,180,180,180,180,180,
1234: 87,181, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1235:
1236: /* block 21 */
1237: 87,182,183,183, 87,184,184,184,184,184,184,184, 87, 87,184,
1238: 184, 87, 87,184,184,184,184,184,184,184,184,184,184,184,184,
1239: 184,184,184,184,184,184,184,184,184, 87,184,184,184,184,184,184,
1240: 184, 87,184,184, 87,184,184,184,184,184, 87, 87,182,184,183,182,
1241: 183,182,182,182,182, 87, 87,183,183, 87, 87,183,183,182, 87, 87,
1242: 87, 87, 87, 87, 87, 87,182,183, 87, 87, 87, 87,184,184, 87,184,
1243: 184,184,182,182, 87, 87,185,185,185,185,185,185,185,185,185,
1244: 186,184, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1245:
1246: /* block 22 */
1247: 87, 87,187,188, 87,188,188,188,188,188,188, 87, 87, 87,188,188,
1248: 188, 87,188,188,188,188, 87, 87, 87,188,188, 87,188, 87,188,188,
1249: 87, 87, 87,188,188, 87, 87, 87,188,188,188, 87, 87, 87,188,188,
1250: 188,188,188,188,188,188,188,188,188,188, 87, 87, 87, 87,189,189,
1251: 187,189,189, 87, 87, 87,189,189,189, 87,189,189,189,187, 87, 87,
1252: 188, 87, 87, 87, 87, 87, 87,189, 87, 87, 87, 87, 87, 87, 87,
1253: 87, 87, 87, 87, 87, 87,190,190,190,190,190,190,190,190,190,
1254: 191,191,191,192,192,192,192,192,192,193,192, 87, 87, 87, 87, 87,
1255:
1256: /* block 23 */
1257: 87,194,194,194, 87,195,195,195,195,195,195,195, 87,195,195,
1258: 195, 87,195,195,195,195,195,195,195,195,195,195,195,195,195,
1259: 195,195,195,195,195,195,195,195,195, 87,195,195,195,195,195,195,
1260: 195,195,195,195, 87,195,195,195,195,195, 87, 87, 87,195,196,196,
1261: 196,194,194,194,194, 87,196,196,196, 87,196,196,196,196, 87, 87,
1262: 87, 87, 87, 87, 87,196,196, 87,195,195, 87, 87, 87, 87, 87, 87,
1263: 195,195,196,196, 87, 87,197,197,197,197,197,197,197,197,197,
1264: 87, 87, 87, 87, 87, 87, 87, 87,198,198,198,198,198,198,198,199,
1265:
1266: /* block 24 */
1267: 87, 87,200,200, 87,201,201,201,201,201,201,201,201, 87,201,201,
1268: 201, 87,201,201,201,201,201,201,201,201,201,201,201,201,201,
1269: 201,201,201,201,201,201,201,201,201, 87,201,201,201,201,201,201,

1270: 201,201,201,201, 87,201,201,201,201,201, 87, 87,202,201,200,202,
1271: 200,200,200,200,200, 87,202,200,200, 87,200,200,202,202, 87, 87,
1272: 87, 87, 87, 87, 87,200,200, 87, 87, 87, 87, 87, 87,201, 87,
1273: 201,201,202,202, 87, 87,203,203,203,203,203,203,203,203,203,
1274: 87, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1275:
1276: /* block 25 */
1277: 87, 87,204,204, 87,205,205,205,205,205,205,205,205, 87,205,205,
1278: 205, 87,205,205,205,205,205,205,205,205,205,205,205,205,205,
1279: 205,205,205,205,205,205,205,205,205, 87,205,205,205,205,205,205,
1280: 205,205,205,205,205,205,205,205,205,205, 87, 87, 87,205,204,204,
1281: 204,206,206,206,206, 87,204,204,204, 87,204,204,204,206, 87, 87,
1282: 87, 87, 87, 87, 87, 87, 87,204, 87, 87, 87, 87, 87, 87, 87,
1283: 205,205,206,206, 87, 87,207,207,207,207,207,207,207,207,207,207,
1284: 208,208,208,208,208,208, 87, 87, 87,209,205,205,205,205,205,205,
1285:
1286: /* block 26 */
1287: 87, 87,210,210, 87,211,211,211,211,211,211,211,211,211,211,
1288: 211,211,211,211,211,211,211, 87, 87, 87,211,211,211,211,211,211,
1289: 211,211,211,211,211,211,211,211,211,211,211,211,211,211,211,
1290: 211,211, 87,211,211,211,211,211,211,211,211,211, 87,211, 87, 87,
1291: 211,211,211,211,211,211,211, 87, 87, 87,212, 87, 87, 87, 87,210,
1292: 210,210,212,212,212, 87,212, 87,210,210,210,210,210,210,210,
1293: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1294: 87, 87,210,210,213, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1295:
1296: /* block 27 */
1297: 87,214,214,214,214,214,214,214,214,214,214,214,214,214,214,
1298: 214,214,214,214,214,214,214,214,214,214,214,214,214,214,214,
1299: 214,214,214,214,214,214,214,214,214,214,214,214,214,214,214,
1300: 214,215,214,214,215,215,215,215,215,215,215,215, 87, 87, 87, 87, 3,
1301: 214,214,214,214,214,214,216,215,215,215,215,215,215,215,215,217,
1302: 218,218,218,218,218,218,218,218,218,217,217, 87, 87, 87, 87,
1303: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1304: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1305:
1306: /* block 28 */
1307: 87,219,219, 87,219, 87, 87,219,219, 87,219, 87, 87,219, 87, 87,
1308: 87, 87, 87, 87,219,219,219,219, 87,219,219,219,219,219,219,
1309: 87,219,219,219, 87,219, 87,219, 87, 87,219,219, 87,219,219,219,
1310: 219,220,219,219,220,220,220,220,220,220, 87,220,220,219, 87, 87,

1311: 219,219,219,219,219, 87,221, 87,220,220,220,220,220,220, 87, 87,
1312: 222,222,222,222,222,222,222,222,222,222, 87, 87,219,219, 87, 87,
1313: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1314: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1315:
1316: /* block 29 */
1317: 223,224,224,224,225,225,225,225,225,225,225,225,225,225,225,
1318: 225,225,225,224,224,224,224,224,226,226,224,224,224,224,224,
1319: 227,227,227,227,227,227,227,227,227,227,228,228,228,228,228,228,
1320: 228,228,228,228,224,226,224,226,224,226,229,230,229,230,231,231,
1321: 223,223,223,223,223,223,223,223, 87,223,223,223,223,223,223,223,
1322: 223,223,223,223,223,223,223,223,223,223,223,223,223,223,223,
1323: 223,223,223,223,223,223,223,223,223,223,223,223, 87, 87, 87,
1324: 87,226,226,226,226,226,226,226,226,226,226,226,226,231,
1325:
1326: /* block 30 */
1327: 226,226,226,226,226,225,226,226,223,223,223,223, 87, 87, 87, 87,
1328: 226,226,226,226,226,226,226,226, 87,226,226,226,226,226,226,226,
1329: 226,226,226,226,226,226,226,226,226,226,226,226,226,226,226,
1330: 226,226,226,226,226,226,226,226,226,226,226,226, 87,224,224,
1331: 224,224,224,224,224,224,226,224,224,224,224,224, 87,224,224,
1332: 225,225,225,225,225, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1333: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1334: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1335:
1336: /* block 31 */
1337: 232,232,232,232,232,232,232,232,232,232,232,232,232,232,232,
1338: 232,232,232,232,232,232,232,232,232,232,232,232,232,232,232,
1339: 232,232,232,232,232,232,232,232,232,232,233,233,234,234,234,
1340: 234,233,234,234,234,234,234,234,233,234,234,233,233,234,234,232,
1341: 235,235,235,235,235,235,235,235,235,235,236,236,236,236,236,
1342: 232,232,232,232,232,232,233,233,234,234,232,232,232,232,234,234,
1343: 234,232,233,233,233,232,232,233,233,233,233,233,233,232,232,
1344: 232,234,234,234,234,232,232,232,232,232,232,232,232,232,232,
1345:
1346: /* block 32 */
1347: 232,232,234,233,233,234,234,233,233,233,233,233,233,234,232,233,
1348: 235,235,235,235,235,235,235,235,235, 87, 87, 87, 87,237,237,
1349: 238,238,238,238,238,238,238,238,238,238,238,238,238,238,238,
1350: 238,238,238,238,238,238,238,238,238,238,238,238,238,238,238,
1351: 238,238,238,238,238,238, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

1352: 239,239,239,239,239,239,239,239,239,239,239,239,239,239,239,
1353: 239,239,239,239,239,239,239,239,239,239,239,239,239,239,239,
1354: 239,239,239,239,239,239,239,239,239,239, 2,240, 87, 87, 87,
1355:
1356: /* block 33 */
1357: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1358: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1359: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1360: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1361: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1362: 241,241,241,241,241,241,241,241,241,241, 87, 87, 87, 87, 87,241,
1363: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1364: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1365:
1366: /* block 34 */
1367: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1368: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1369: 241,241,241, 87, 87, 87, 87, 87,241,241,241,241,241,241,241,
1370: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1371: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1372: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1373: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1374: 241,241,241,241,241,241,241,241,241,241, 87, 87, 87, 87, 87, 87,
1375:
1376: /* block 35 */
1377: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1378: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1379: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1380: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1381: 242,242,242,242,242,242,242,242,242, 87,242,242,242,242, 87, 87,
1382: 242,242,242,242,242,242,242, 87,242, 87,242,242,242,242, 87, 87,
1383: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1384: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1385:
1386: /* block 36 */
1387: 242,242,242,242,242,242,242,242, 87,242,242,242,242, 87, 87,
1388: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1389: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1390: 242, 87,242,242,242,242, 87, 87,242,242,242,242,242,242, 87,
1391: 242, 87,242,242,242,242, 87, 87,242,242,242,242,242,242,242,
1392: 242,242,242,242,242,242,242, 87,242,242,242,242,242,242,242,

1393: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1394: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1395:
1396: /* block 37 */
1397: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1398: 242, 87,242,242,242,242, 87, 87,242,242,242,242,242,242,242,242,
1399: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1400: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1401: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1402: 242,242,242,242,242,242,242,242,242,242,242, 87, 87, 87, 87,243,
1403: 244,245,245,245,245,245,245,245,245,246,246,246,246,246,246,246,
1404: 246,246,246,246,246,246,246,246,246,246,246,246, 87, 87, 87,
1405:
1406: /* block 38 */
1407: 242,242,242,242,242,242,242,242,242,242,242,242,242,242,242,
1408: 244,244,244,244,244,244,244,244,244, 87, 87, 87, 87, 87, 87,
1409: 247,247,247,247,247,247,247,247,247,247,247,247,247,247,247,
1410: 247,247,247,247,247,247,247,247,247,247,247,247,247,247,247,
1411: 247,247,247,247,247,247,247,247,247,247,247,247,247,247,247,
1412: 247,247,247,247,247,247,247,247,247,247,247,247,247,247,247,
1413: 247,247,247,247,247,247,247,247,247,247,247,247,247,247,247,
1414: 247,247,247,247,247, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1415:
1416: /* block 39 */
1417: 87,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1418: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1419: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1420: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1421: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1422: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1423: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1424: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1425:
1426: /* block 40 */
1427: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1428: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1429: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1430: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1431: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1432: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1433: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,248,

1434: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1435:
1436: /* block 41 */
1437: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1438: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1439: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1440: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1441: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1442: 248,248,248,248,248,248,248,248,248,248,248,248,248,248,
1443: 248,248,248,248,248,248,248,248,248,248,248,248,249,249,248,
1444: 248,248,248,248,248,248,248, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1445:
1446: /* block 42 */
1447: 250,251,251,251,251,251,251,251,251,251,251,251,251,251,
1448: 251,251,251,251,251,251,251,251,251,251,252,253, 87, 87, 87,
1449: 254,254,254,254,254,254,254,254,254,254,254,254,254,254,
1450: 254,254,254,254,254,254,254,254,254,254,254,254,254,254,
1451: 254,254,254,254,254,254,254,254,254,254,254,254,254,254,
1452: 254,254,254,254,254,254,254,254,254,254,254,254,254,254,
1453: 254,254,254,254,254,254,254,254,254,254, 2, 2, 2,255,255,
1454: 255, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1455:
1456: /* block 43 */
1457: 256,256,256,256,256,256,256,256,256,256,256,256, 87,256,256,
1458: 256,256,257,257,257, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1459: 258,258,258,258,258,258,258,258,258,258,258,258,258,258,
1460: 258,258,259,259,259, 2, 2, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1461: 260,260,260,260,260,260,260,260,260,260,260,260,260,260,
1462: 260,260,261,261, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1463: 262,262,262,262,262,262,262,262,262,262,262,262, 87,262,262,
1464: 262, 87,263,263, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1465:
1466: /* block 44 */
1467: 264,264,264,264,264,264,264,264,264,264,264,264,264,264,
1468: 264,264,264,264,264,264,264,264,264,264,264,264,264,264,
1469: 264,264,264,264,264,264,264,264,264,264,264,264,264,264,
1470: 264,264,264,264,265,265,266,267,267,267,267,267,267,266,266,
1471: 266,266,266,266,266,266,267,266,266,267,267,267,267,267,
1472: 267,267,267,267,268,268,268,269,268,268,268,270,264,267, 87, 87,
1473: 271,271,271,271,271,271,271,271,271,271, 87, 87, 87, 87, 87, 87,
1474: 272,272,272,272,272,272,272,272,272,272, 87, 87, 87, 87, 87, 87,

1475:
1476: /* block 45 */
1477: 273,273, 2, 2,273, 2,274,273,273,273,275,275,275,276, 87,
1478: 277,277,277,277,277,277,277,277,277,277, 87, 87, 87, 87, 87, 87,
1479: 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,
1480: 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,
1481: 278,278,278,279,278,278,278,278,278,278,278,278,278,278,278,
1482: 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,
1483: 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,
1484: 278,278,278,278,278,278,278,278, 87, 87, 87, 87, 87, 87, 87,
1485:
1486: /* block 46 */
1487: 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,
1488: 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,
1489: 278,278,278,278,278,278,278,278,278,275,278, 87, 87, 87, 87, 87,
1490: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1491: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1492: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1493: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1494: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1495:
1496: /* block 47 */
1497: 280,280,280,280,280,280,280,280,280,280,280,280,280,280,280,
1498: 280,280,280,280,280,280,280,280,280,280,280,280, 87, 87, 87,
1499: 281,281,281,282,282,282,282,281,281,282,282,282, 87, 87, 87, 87,
1500: 282,282,281,282,282,282,282,282,282,281,281,281, 87, 87, 87, 87,
1501: 283, 87, 87, 87,284,284,285,285,285,285,285,285,285,285,285,
1502: 286,286,286,286,286,286,286,286,286,286,286,286,286,286,286,
1503: 286,286,286,286,286,286,286,286,286,286,286,286,286,286, 87, 87,
1504: 286,286,286,286,286, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1505:
1506: /* block 48 */
1507: 287,287,287,287,287,287,287,287,287,287,287,287,287,287,287,
1508: 287,287,287,287,287,287,287,287,287,287,287,287,287,287,287,
1509: 287,287,287,287,287,287,287,287,287,287, 87, 87, 87, 87, 87, 87,
1510: 288,288,288,288,288,288,288,288,288,288,288,288,288,288,288,
1511: 288,287,287,287,287,287,287,287,288,288, 87, 87, 87, 87, 87, 87,
1512: 289,289,289,289,289,289,289,289,289,289, 87, 87, 87, 87,290,290,
1513: 291,291,291,291,291,291,291,291,291,291,291,291,291,291,291,
1514: 291,291,291,291,291,291,291,291,291,291,291,291,291,291,291,
1515:

1516: /* block 49 */
1517: 292,292,292,292,292,292,292,292,292,292,292,292,292,292,292,
1518: 292,292,292,292,292,292,292,293,293,294,294,294, 87, 87,295,295,
1519: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1520: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1521: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1522: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1523: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1524: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1525:
1526: /* block 50 */
1527: 296,296,296,296,297,298,298,298,298,298,298,298,298,298,298,
1528: 298,298,298,298,298,298,298,298,298,298,298,298,298,298,298,
1529: 298,298,298,298,298,298,298,298,298,298,298,298,298,298,298,
1530: 298,298,298,298,296,297,296,296,296,296,297,296,297,297,297,
1531: 297,297,296,297,297,298,298,298,298,298,298, 87, 87, 87, 87,
1532: 299,299,299,299,299,299,299,299,299,299,300,300,300,300,300,300,
1533: 300,301,301,301,301,301,301,301,301,301,301,296,296,296,296,296,
1534: 296,296,296,296,301,301,301,301,301,301,301,301,301, 87, 87, 87,
1535:
1536: /* block 51 */
1537: 302,302,303,304,304,304,304,304,304,304,304,304,304,304,304,
1538: 304,304,304,304,304,304,304,304,304,304,304,304,304,304,304,
1539: 304,303,302,302,302,302,303,303,302,302,303, 87, 87, 87,304,304,
1540: 305,305,305,305,305,305,305,305,305,305, 87, 87, 87, 87, 87, 87,
1541: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1542: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1543: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1544: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1545:
1546: /* block 52 */
1547: 306,306,306,306,306,306,306,306,306,306,306,306,306,306,306,
1548: 306,306,306,306,306,306,306,306,306,306,306,306,306,306,306,
1549: 306,306,306,306,307,307,307,307,307,307,307,307,308,308,308,308,
1550: 308,308,308,308,307,307,308,308, 87, 87, 87,309,309,309,309,309,
1551: 310,310,310,310,310,310,310,310,310,310, 87, 87, 87,306,306,306,
1552: 311,311,311,311,311,311,311,311,311,311,312,312,312,312,312,312,
1553: 312,312,312,312,312,312,312,312,312,312,312,312,312,312,312,
1554: 312,312,312,312,312,312,312,312,313,313,313,313,313,313,314,314,
1555:
1556: /* block 53 */

1557: 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
 1558: 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
 1559: 14, 14, 14, 14, 14, 14, 94, 94, 94, 94, 94, 315, 80, 80, 80, 80,
 1560: 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
 1561: 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
 1562: 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 88, 88, 88,
 1563: 88, 88, 14, 14, 14, 14, 94, 94, 94, 94, 94, 14, 14, 14, 14, 14,
 1564: 14, 14, 14, 14, 14, 14, 14, 14, 316, 317, 14, 14, 14, 318, 14, 14,
 1565:
 1566: /* block 54 */
 1567: 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
 1568: 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 80, 80, 80, 80, 80,
 1569: 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80,
 1570: 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 88,
 1571: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
 1572: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
 1573: 82, 82, 82, 82, 82, 82, 82, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1574: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 82, 82,
 1575:
 1576: /* block 55 */
 1577: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1578: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1579: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1580: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1581: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1582: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1583: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1584: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1585:
 1586: /* block 56 */
 1587: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1588: 21, 22, 21, 22, 21, 22, 14, 14, 14, 14, 14, 319, 14, 14, 320, 14,
 1589: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1590: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1591: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1592: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1593: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1594: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
 1595:
 1596: /* block 57 */
 1597: 321, 321, 321, 321, 321, 321, 321, 321, 322, 322, 322, 322, 322, 322, 322, 322,

1598: 321,321,321,321,321,321, 87, 87,322,322,322,322,322,322, 87, 87,
1599: 321,321,321,321,321,321,321,321,321,322,322,322,322,322,322,322,
1600: 321,321,321,321,321,321,321,321,321,322,322,322,322,322,322,322,
1601: 321,321,321,321,321,321, 87, 87,322,322,322,322,322,322, 87, 87,
1602: 94,321, 94,321, 94,321, 94,321, 87,322, 87,322, 87,322, 87,322,
1603: 321,321,321,321,321,321,321,321,322,322,322,322,322,322,322,
1604: 323,323,324,324,324,324,325,325,326,326,327,327,328,328, 87, 87,
1605:
1606: /* block 58 */
1607: 321,321,321,321,321,321,321,321,329,329,329,329,329,329,329,
1608: 321,321,321,321,321,321,321,321,329,329,329,329,329,329,329,
1609: 321,321,321,321,321,321,321,321,329,329,329,329,329,329,329,
1610: 321,321, 94,330, 94, 87, 94, 94,322,322,331,331,332, 86,333, 86,
1611: 86, 86, 94,330, 94, 87, 94, 94,334,334,334,334,332, 86, 86, 86,
1612: 321,321, 94, 94, 87, 87, 94, 94,322,322,335,335, 87, 86, 86, 86,
1613: 321,321, 94, 94, 94,113, 94, 94,322,322,336,336,117, 86, 86, 86,
1614: 87, 87, 94,330, 94, 87, 94, 94,337,337,338,338,332, 86, 86, 87,
1615:
1616: /* block 59 */
1617: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 16,339,339, 16, 16,
1618: 7, 7, 7, 7, 7, 7, 2, 2, 15, 19, 4, 15, 15, 19, 4, 15,
1619: 2, 2, 2, 2, 2, 2, 2, 2, 2,340,341, 16, 16, 16, 16, 16, 1,
1620: 2, 2, 2, 2, 2, 2, 2, 2, 2, 15, 19, 2, 2, 2, 2, 11,
1621: 11, 2, 2, 2, 6, 4, 5, 2, 2, 2, 2, 2, 2, 2, 2, 2,
1622: 2, 2, 6, 2, 11, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1,
1623: 16, 16, 16, 16, 16, 87, 87, 87, 87, 87, 16, 16, 16, 16, 16, 16,
1624: 17, 14, 87, 87, 17, 17, 17, 17, 17, 17, 6, 6, 6, 4, 5, 14,
1625:
1626: /* block 60 */
1627: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 6, 6, 6, 4, 5, 87,
1628: 80, 80, 80, 80, 80, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1629: 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
1630: 3, 3, 3, 3, 3, 3, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1631: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1632: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
1633: 342, 82, 342, 342, 342, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
1634: 82, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1635:
1636: /* block 61 */
1637: 13, 13, 343, 13, 13, 13, 13, 343, 13, 13, 344, 343, 343, 343, 344, 344,
1638: 343, 343, 343, 344, 13, 343, 13, 13, 13, 343, 343, 343, 343, 343, 13, 13,

1639: 13, 13, 13, 13,343, 13,345, 13,343, 13,346,347,343,343, 13,344,
 1640: 343,343,348,343,344,349,349,349,349,344, 13, 13,344,344,343,343,
 1641: 6, 6, 6, 6, 6,343,344,344,344,344, 13, 6, 13, 13,350, 13,
 1642: 87, 87, 87, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
 1643: 351,351,351,351,351,351,351,351,351,351,351,351,351,351,351,351,
 1644: 352,352,352,352,352,352,352,352,352,352,352,352,352,352,352,352,
 1645:
 1646: /* block 62 */
 1647: 353,353,353, 21, 22,353,353,353,353, 87, 87, 87, 87, 87, 87, 87,
 1648: 6, 6, 6, 6, 6, 13, 13, 13, 13, 13, 6, 6, 13, 13, 13, 13,
 1649: 6, 13, 13, 6, 13, 13, 6, 13, 13, 13, 13, 13, 13, 13, 6, 13,
 1650: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1651: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 6, 6,
 1652: 13, 13, 6, 13, 6, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1653: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1654: 13, 13, 13, 13, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1655:
 1656: /* block 63 */
 1657: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1658: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1659: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1660: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1661: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1662: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1663: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1664: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1665:
 1666: /* block 64 */
 1667: 13, 13, 13, 13, 13, 13, 13, 13, 6, 6, 6, 6, 13, 13, 13, 13,
 1668: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1669: 6, 6, 13, 13, 13, 13, 13, 13, 13, 13, 4, 5, 13, 13, 13, 13, 13,
 1670: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1671: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1672: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1673: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1674: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 6, 13, 13, 13,
 1675:
 1676: /* block 65 */
 1677: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1678: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 6, 6, 6, 6, 6,
 1679: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,

1680: 6, 6, 6, 6, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1681: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1682: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 6, 6, 6, 6,
1683: 6, 6, 13, 13, 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87,
1684: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1685:
1686: /* block 66 */
1687: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1688: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1689: 13, 13, 13, 13, 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87,
1690: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1691: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 87, 87, 87, 87,
1692: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1693: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
1694: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
1695:
1696: /* block 67 */
1697: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
1698: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 13, 13, 13, 13,
1699: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1700: 13, 13, 13, 13, 13, 13, 354, 354, 354, 354, 354, 354, 354, 354, 354,
1701: 354, 354, 354, 354, 354, 354, 354, 354, 354, 354, 354, 354, 354, 354,
1702: 355, 355, 355, 355, 355, 355, 355, 355, 355, 355, 355, 355, 355, 355,
1703: 355, 355, 355, 355, 355, 355, 355, 355, 355, 17, 17, 17, 17, 17, 17,
1704: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
1705:
1706: /* block 68 */
1707: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1708: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1709: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1710: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1711: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1712: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1713: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1714: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1715:
1716: /* block 69 */
1717: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1718: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1719: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1720: 13, 13, 13, 13, 13, 13, 6, 13, 13, 13, 13, 13, 13, 13, 13,

1721: 13, 6, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1722: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1723: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1724: 13, 13, 13, 13, 13, 13, 13, 13, 13, 6, 6, 6, 6, 6, 6, 6, 6,
 1725:
 1726: /* block 70 */
 1727: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1728: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1729: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1730: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1731: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1732: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1733: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 6,
 1734: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1735:
 1736: /* block 71 */
 1737: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1738: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 87,
 1739: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1740: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 87, 87,
 1741: 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1742: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1743: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1744: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1745:
 1746: /* block 72 */
 1747: 87, 13, 13, 13, 13, 13, 87, 13, 13, 13, 13, 87, 87, 13, 13, 13, 13,
 1748: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1749: 13, 13, 13, 13, 13, 13, 13, 13, 87, 13, 13, 13, 13, 13, 13, 13,
 1750: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1751: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 13, 87, 13,
 1752: 13, 13, 13, 87, 87, 87, 13, 87, 13, 13, 13, 13, 13, 13, 13, 87,
 1753: 87, 13, 13, 13, 13, 13, 13, 13, 4, 5, 4, 5, 4, 5, 4, 5,
 1754: 4, 5, 4, 5, 4, 5, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
 1755:
 1756: /* block 73 */
 1757: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
 1758: 17, 17, 17, 17, 13, 87, 87, 87, 13, 13, 13, 13, 13, 13, 13, 13,
 1759: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1760: 87, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87,
 1761: 6, 6, 6, 6, 6, 4, 5, 6, 6, 6, 6, 87, 6, 87, 87, 87,

1762: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1763: 6, 6, 6, 6, 6, 6, 4, 5, 4, 5, 4, 5, 4, 5, 4, 5,
 1764: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1765:
 1766: /* block 74 */
 1767: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1768: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1769: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1770: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1771: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1772: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1773: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1774: 356,356,356,356,356,356,356,356,356,356,356,356,356,356,
 1775:
 1776: /* block 75 */
 1777: 6, 6, 6, 4, 5, 4, 5, 4, 5, 4, 5, 4, 5, 4, 5, 4,
 1778: 5, 4, 5, 4, 5, 4, 5, 4, 5, 6, 6, 6, 6, 6, 6, 6,
 1779: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1780: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1781: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1782: 6, 6, 6, 6, 6, 6, 6, 6, 4, 5, 4, 5, 6, 6, 6, 6,
 1783: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1784: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 4, 5, 6, 6,
 1785:
 1786: /* block 76 */
 1787: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1788: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1789: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
 1790: 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
 1791: 6, 6, 6, 6, 6, 13, 13, 6, 6, 6, 6, 6, 6, 87, 87, 87,
 1792: 13, 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1793: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1794: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
 1795:
 1796: /* block 77 */
 1797: 357,357,357,357,357,357,357,357,357,357,357,357,357,357,
 1798: 357,357,357,357,357,357,357,357,357,357,357,357,357,357,
 1799: 357,357,357,357,357,357,357,357,357,357,357,357,357, 87,
 1800: 358,358,358,358,358,358,358,358,358,358,358,358,358,358,
 1801: 358,358,358,358,358,358,358,358,358,358,358,358,358,358,
 1802: 358,358,358,358,358,358,358,358,358,358,358,358,358, 87,

1803: 21, 22, 359, 360, 361, 362, 363, 21, 22, 21, 22, 21, 22, 364, 365, 366,
1804: 87, 14, 21, 22, 14, 21, 22, 14, 14, 14, 14, 14, 80, 87, 87,
1805:
1806: /* block 78 */
1807: 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110,
1808: 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110,
1809: 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110,
1810: 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110,
1811: 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110,
1812: 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110, 109, 110,
1813: 109, 110, 109, 110, 367, 368, 368, 368, 368, 368, 368, 87, 87, 87, 87, 87,
1814: 87, 87, 87, 87, 87, 87, 87, 87, 87, 369, 369, 369, 369, 370, 369, 369,
1815:
1816: /* block 79 */
1817: 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371,
1818: 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371, 371,
1819: 371, 371, 371, 371, 371, 371, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1820: 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372,
1821: 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372,
1822: 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372, 372,
1823: 372, 372, 372, 372, 372, 372, 87, 87, 87, 87, 87, 87, 87, 87, 87, 373,
1824: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1825:
1826: /* block 80 */
1827: 242, 242, 242, 242, 242, 242, 242, 242, 242, 242, 242, 242, 242, 242, 242,
1828: 242, 242, 242, 242, 242, 242, 242, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1829: 242, 242, 242, 242, 242, 242, 242, 87, 242, 242, 242, 242, 242, 242, 242, 87,
1830: 242, 242, 242, 242, 242, 242, 242, 87, 242, 242, 242, 242, 242, 242, 242, 87,
1831: 242, 242, 242, 242, 242, 242, 242, 87, 242, 242, 242, 242, 242, 242, 242, 87,
1832: 242, 242, 242, 242, 242, 242, 242, 87, 242, 242, 242, 242, 242, 242, 242, 87,
1833: 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126,
1834: 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126,
1835:
1836: /* block 81 */
1837: 2, 2, 15, 19, 15, 19, 2, 2, 2, 15, 19, 2, 15, 19, 2, 2,
1838: 2, 2, 2, 2, 2, 2, 2, 7, 2, 2, 7, 2, 15, 19, 2, 2,
1839: 15, 19, 4, 5, 4, 5, 4, 5, 4, 5, 2, 2, 2, 2, 2, 81,
1840: 2, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1841: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1842: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1843: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

1844: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1845:
1846: /* block 82 */
1847: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1848: 374,374,374,374,374,374,374,374,374, 87,374,374,374,374,374,
1849: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1850: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1851: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1852: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1853: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1854: 374,374,374,374, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1855:
1856: /* block 83 */
1857: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1858: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1859: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1860: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1861: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1862: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1863: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1864: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1865:
1866: /* block 84 */
1867: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1868: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1869: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1870: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1871: 374,374,374,374,374,374,374,374,374,374,374,374,374,374,374,
1872: 374,374,374,374,374,374, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1873: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1874: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 87, 87, 87,
1875:
1876: /* block 85 */
1877: 1, 2, 2, 2, 13,375,349,376, 4, 5, 4, 5, 4, 5, 4, 5,
1878: 4, 5, 13, 13, 4, 5, 4, 5, 4, 5, 4, 5, 7, 4, 5, 5,
1879: 13,376,376,376,376,376,376,376,376,376, 82, 82, 82, 82, 82, 82,
1880: 7, 81, 81, 81, 81, 81, 13, 13,376,376,376,375,349, 2, 13, 13,
1881: 87,377,377,377,377,377,377,377,377,377,377,377,377,377,377,
1882: 377,377,377,377,377,377,377,377,377,377,377,377,377,377,377,
1883: 377,377,377,377,377,377,377,377,377,377,377,377,377,377,377,
1884: 377,377,377,377,377,377,377,377,377,377,377,377,377,377,377,

1885:
1886: /* block 86 */
1887: 377,377,377,377,377,377,377,377,377,377,377,377,377,377,377,
1888: 377,377,377,377,377,377,377, 87, 87, 82, 82, 10, 10,378,378,377,
1889: 7,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
1890: 379,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
1891: 379,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
1892: 379,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
1893: 379,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
1894: 379,379,379,379,379,379,379,379,379,379, 2, 81,380,380,379,
1895:
1896: /* block 87 */
1897: 87, 87, 87, 87, 87,381,381,381,381,381,381,381,381,381,381,381,
1898: 381,381,381,381,381,381,381,381,381,381,381,381,381,381,381,
1899: 381,381,381,381,381,381,381,381,381,381,381,381,381, 87, 87,
1900: 87,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1901: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1902: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1903: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1904: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
1905:
1906: /* block 88 */
1907: 241,241,241,241,241,241,241,241,241,241,241,241,241, 87,
1908: 13, 13, 17, 17, 17, 17, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1909: 381,381,381,381,381,381,381,381,381,381,381,381,381,381,381,
1910: 381,381,381,381,381,381,381,381, 87, 87, 87, 87, 87, 87, 87, 87,
1911: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1912: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1913: 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1914: 379,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
1915:
1916: /* block 89 */
1917: 382,382,382,382,382,382,382,382,382,382,382,382,382,382,382,
1918: 382,382,382,382,382,382,382,382,382,382,382,382,382,382, 87,
1919: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 13, 13, 13, 13, 13,
1920: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1921: 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1922: 13, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
1923: 382,382,382,382,382,382,382,382,382,382,382,382,382,382,382,
1924: 382,382,382,382,382,382,382,382,382,382,382,382,382,382, 13,
1925:

1926: /* block 90 */
1927: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 13, 13, 13, 13, 13, 13,
1928: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1929: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1930: 13, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
1931: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1932: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,
1933: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,
1934: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383, 87,
1935:
1936: /* block 91 */
1937: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,
1938: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,
1939: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,
1940: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,
1941: 383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,383,
1942: 383,383,383,383,383,383,383,383, 13, 13, 13, 13, 13, 13, 13, 13,
1943: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1944: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1945:
1946: /* block 92 */
1947: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1948: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1949: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1950: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1951: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1952: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1953: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1954: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1955:
1956: /* block 93 */
1957: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1958: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1959: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1960: 384,384,384,384,384,384, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1961: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1962: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1963: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1964: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
1965:
1966: /* block 94 */

1967: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1968: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1969: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1970: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
1971: 384,384,384,384, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1972: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1973: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1974: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
1975:
1976: /* block 95 */
1977: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1978: 385,385,385,385,385,386,385,385,385,385,385,385,385,385,385,385,
1979: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1980: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1981: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1982: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1983: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1984: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1985:
1986: /* block 96 */
1987: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1988: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1989: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1990: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1991: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1992: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1993: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1994: 385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,385,
1995:
1996: /* block 97 */
1997: 385,385,385,385,385,385,385,385,385,385,385, 87, 87, 87,
1998: 387,387,387,387,387,387,387,387,387,387,387,387,387,387,387,387,
1999: 387,387,387,387,387,387,387,387,387,387,387,387,387,387,387,387,
2000: 387,387,387,387,387,387,387,387,387,387,387,387,387,387,387,387,
2001: 387,387,387,387,387,387,387, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2002: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2003: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2004: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2005:
2006: /* block 98 */
2007: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,

2008: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2009: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2010: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2011: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2012: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2013: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2014: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2015:
2016: /* block 99 */
2017: 388,388,388,388,388,388,388,388,388,388,388,388,388,389,390,390,390,
2018: 388,388,388,388,388,388,388,388,388,388,388,388,388,388,388,
2019: 391,391,391,391,391,391,391,391,391,391,388,388, 87, 87, 87, 87,
2020: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2021: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,124,
2022: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,124,
2023: 87, 87,123,124,123,124,123,124,123,124,123,124,123,124,392,126,
2024: 127,127,127,393, 87, 87, 87, 87, 87, 87, 87, 87, 87,126,126,393,316,
2025:
2026: /* block 100 */
2027: 123,124,123,124,123,124,123,124,123,124,123,124,123,124,123,124,
2028: 123,124,123,124,123,124,123,124, 87, 87, 87, 87, 87, 87, 87, 87,
2029: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2030: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2031: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2032: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2033: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2034: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2035:
2036: /* block 101 */
2037: 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
2038: 10, 10, 10, 10, 10, 10, 10, 81, 81, 81, 81, 81, 81, 81, 81,
2039: 10, 10, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
2040: 14, 14, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
2041: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
2042: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
2043: 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22, 21, 22,
2044: 80, 14, 14, 14, 14, 14, 14, 14, 14, 21, 22, 21, 22,394, 21, 22,
2045:
2046: /* block 102 */
2047: 21, 22, 21, 22, 21, 22, 21, 22, 81, 10, 10, 21, 22, 87, 87, 87,
2048: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

2049: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2050: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2051: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2052: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2053: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2054: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 45, 45, 45, 45, 45,
2055:
2056: /* block 103 */
2057: 395,395,396,395,395,395,396,395,395,395,395,396,395,395,395,395,
2058: 395,395,395,395,395,395,395,395,395,395,395,395,395,395,395,
2059: 395,395,395,397,397,396,396,397,398,398,398,398, 87, 87, 87, 87,
2060: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2061: 399,399,399,399,399,399,399,399,399,399,399,399,399,399,399,399,
2062: 399,399,399,399,399,399,399,399,399,399,399,399,399,399,399,
2063: 399,399,399,399,399,399,399,399,399,399,399,399,399,399,399,
2064: 399,399,399,399,400,400,400,400, 87, 87, 87, 87, 87, 87, 87, 87,
2065:
2066: /* block 104 */
2067: 401,401,402,402,402,402,402,402,402,402,402,402,402,402,402,
2068: 402,402,402,402,402,402,402,402,402,402,402,402,402,402,402,
2069: 402,402,402,402,402,402,402,402,402,402,402,402,402,402,402,
2070: 402,402,402,402,401,401,401,401,401,401,401,401,401,401,401,
2071: 401,401,401,401,403, 87, 87, 87, 87, 87, 87, 87, 87, 87,404,404,
2072: 405,405,405,405,405,405,405,405,405,405, 87, 87, 87, 87, 87, 87,
2073: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2074: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2075:
2076: /* block 105 */
2077: 406,406,406,406,406,406,406,406,406,406,407,407,407,407,407,407,
2078: 407,407,407,407,407,407,407,407,407,407,407,407,407,407,407,
2079: 407,407,407,407,407,407,408,408,408,408,408,408,408,408,409,409,
2080: 410,410,410,410,410,410,410,410,410,410,410,410,410,410,410,
2081: 410,410,410,410,410,410,410,411,411,411,411,411,411,411,411,411,
2082: 411,411,412,412, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,413,
2083: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2084: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2085:
2086: /* block 106 */
2087: 414,414,414,414,414,414,414,414,414,414,414,414,414,414,414,
2088: 414,414,414,414,414,414,414,414,414,414,414,414,414,414,414,
2089: 414,414,414,414,414,414,414,414,415,415,415,415,415,415,416,

2090: 416,415,415,416,416,415,415, 87, 87, 87, 87, 87, 87, 87, 87,
2091: 414,414,414,415,414,414,414,414,414,414,415,416, 87, 87,
2092: 417,417,417,417,417,417,417,417,417, 87, 87,418,418,418,418,
2093: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2094: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2095:
2096: /* block 107 */
2097: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2098: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2099: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2100: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2101: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2102: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2103: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2104: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2105:
2106: /* block 108 */
2107: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2108: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2109: 241,241,241,241, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2110: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2111: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2112: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2113: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2114: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2115:
2116: /* block 109 */
2117: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2118: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2119: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2120: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2121: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2122: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2123: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2124: 419,419,419,419,419,419,419,419,419,419,419,419,419,419,
2125:
2126: /* block 110 */
2127: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2128: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2129: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2130: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,

2131: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2132: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2133: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2134: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2135:
2136: /* block 111 */
2137: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2138: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2139: 384,384,384,384,384,384,384,384,384,384,384,384,384, 87, 87,
2140: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2141: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2142: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2143: 384,384,384,384,384,384,384,384,384,384,384, 87, 87, 87, 87, 87,
2144: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2145:
2146: /* block 112 */
2147: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2148: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2149: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2150: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2151: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2152: 384,384,384,384,384,384,384,384,384,384, 87, 87, 87, 87, 87, 87,
2153: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2154: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2155:
2156: /* block 113 */
2157: 14, 14, 14, 14, 14, 14, 14, 87, 87, 87, 87, 87, 87, 87, 87,
2158: 87, 87, 87,134,134,134,134,134, 87, 87, 87, 87, 87,139,136,139,
2159: 139,139,139,139,139,139,139,139,139,139,421,139,139,139,139,139,
2160: 139,139,139,139,139,139,139, 87,139,139,139,139,139, 87,139, 87,
2161: 139,139, 87,139,139, 87,139,139,139,139,139,139,139,139,139,
2162: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2163: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2164: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2165:
2166: /* block 114 */
2167: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2168: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2169: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2170: 145,145, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2171: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

2172: 87, 87, 87, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2173: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2174: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2175:
2176: /* block 115 */
2177: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2178: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2179: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2180: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2181: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2182: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2183: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2184: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2185:
2186: /* block 116 */
2187: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2188: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2189: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2190: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 4, 5,
2191: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2192: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2193: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2194: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2195:
2196: /* block 117 */
2197: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2198: 87, 87, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2199: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2200: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
2201: 145, 145, 145, 145, 145, 145, 145, 145, 87, 87, 87, 87, 87, 87, 87, 87,
2202: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2203: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2204: 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 142, 13, 87, 87,
2205:
2206: /* block 118 */
2207: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2208: 2, 2, 2, 2, 2, 2, 2, 4, 5, 2, 87, 87, 87, 87, 87, 87,
2209: 82, 82, 82, 82, 82, 82, 82, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2210: 2, 7, 7, 11, 11, 4, 5, 4, 5, 4, 5, 4, 5, 4, 5, 4,
2211: 5, 4, 5, 4, 5, 2, 2, 4, 5, 2, 2, 2, 2, 11, 11, 11,
2212: 2, 2, 2, 87, 2, 2, 2, 2, 7, 4, 5, 4, 5, 4, 5, 2,

2213: 2, 2, 6, 7, 6, 6, 6, 87, 2, 3, 2, 2, 87, 87, 87, 87,
2214: 145,145,145,145,145, 87,145,145,145,145,145,145,145,145,145,
2215:
2216: /* block 119 */
2217: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2218: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2219: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2220: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2221: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2222: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2223: 145,145,145,145,145,145,145,145,145,145,145,145,145,145,
2224: 145,145,145,145,145,145,145,145,145,145,145,145,145, 87, 87, 16,
2225:
2226: /* block 120 */
2227: 87, 2, 2, 2, 3, 2, 2, 2, 4, 5, 2, 6, 2, 7, 2, 2,
2228: 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 2, 2, 6, 6, 6, 2,
2229: 2, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
2230: 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 4, 2, 5, 10, 11,
2231: 10, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
2232: 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 4, 6, 5, 6, 4,
2233: 5, 2, 4, 5, 2, 2,379,379,379,379,379,379,379,379,379,379,
2234: 81,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
2235:
2236: /* block 121 */
2237: 379,379,379,379,379,379,379,379,379,379,379,379,379,379,379,
2238: 379,379,379,379,379,379,379,379,379,379,379,379,379, 81, 81,
2239: 241,241,241,241,241,241,241,241,241,241,241,241,241,241,241,
2240: 241,241,241,241,241,241,241,241,241,241,241,241,241,241, 87,
2241: 87, 87,241,241,241,241,241,241, 87, 87,241,241,241,241,241,
2242: 87, 87,241,241,241,241,241,241, 87, 87,241,241,241, 87, 87, 87,
2243: 3, 3, 6, 10, 13, 3, 3, 87, 13, 6, 6, 6, 6, 13, 13, 87,
2244: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 16, 16, 16, 13, 13, 87, 87,
2245:
2246: /* block 122 */
2247: 422,422,422,422,422,422,422,422,422,422,422, 87,422,422,422,
2248: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2249: 422,422,422,422,422,422,422, 87,422,422,422,422,422,422,422,
2250: 422,422,422,422,422,422,422,422,422,422, 87,422,422, 87,422,
2251: 422,422,422,422,422,422,422,422,422,422,422,422, 87, 87,
2252: 422,422,422,422,422,422,422,422,422,422,422,422, 87, 87,
2253: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

2254: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2255:
2256: /* block 123 */
2257: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2258: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2259: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2260: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2261: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2262: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2263: 422,422,422,422,422,422,422,422,422,422,422,422,422,422,422,
2264: 422,422,422,422,422,422,422,422,422,422,422,422, 87, 87, 87, 87, 87,
2265:
2266: /* block 124 */
2267: 2, 2, 13, 87, 87, 87, 87, 17, 17, 17, 17, 17, 17, 17, 17, 17,
2268: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
2269: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
2270: 17, 17, 17, 17, 87, 87, 87, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2271: 423,423,423,423,423,423,423,423,423,423,423,423,423,423,423,
2272: 423,423,423,423,423,423,423,423,423,423,423,423,423,423,423,
2273: 423,423,423,423,423,423,423,423,423,423,423,423,423,423,423,
2274: 423,423,423,423,423,424,424,424,424,425,425,425,425,425,425,
2275:
2276: /* block 125 */
2277: 425,425,425,425,425,425,425,425,425,424, 87, 87, 87, 87, 87,
2278: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 87, 87, 87,
2279: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2280: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2281: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2282: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2283: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2284: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 82, 87, 87,
2285:
2286: /* block 126 */
2287: 426,426,426,426,426,426,426,426,426,426,426,426,426,426,426,
2288: 426,426,426,426,426,426,426,426,426,426,426,426, 87, 87, 87,
2289: 427,427,427,427,427,427,427,427,427,427,427,427,427,427,427,
2290: 427,427,427,427,427,427,427,427,427,427,427,427,427,427,427,
2291: 427,427,427,427,427,427,427,427,427,427,427,427,427,427,427,
2292: 427, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2293: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2294: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

2295:
2296: /* block 127 */
2297: 428,428,428,428,428,428,428,428,428,428,428,428,428,428,428,428,
2298: 428,428,428,428,428,428,428,428,428,428,428,428,428,428, 87,
2299: 429,429,429,429, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2300: 430,430,430,430,430,430,430,430,430,430,430,430,430,430,430,
2301: 430,431,430,430,430,430,430,430,430,430,431, 87, 87, 87, 87, 87,
2302: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2303: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2304: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2305:
2306: /* block 128 */
2307: 432,432,432,432,432,432,432,432,432,432,432,432,432,432,432,
2308: 432,432,432,432,432,432,432,432,432,432,432,432,432, 87,433,
2309: 434,434,434,434,434,434,434,434,434,434,434,434,434,434,434,
2310: 434,434,434,434,434,434,434,434,434,434,434,434,434,434,434,
2311: 434,434,434,434, 87, 87, 87, 87,434,434,434,434,434,434,434,434,
2312: 435,436,436,436,436,436, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2313: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2314: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2315:
2316: /* block 129 */
2317: 437,437,437,437,437,437,437,437,437,437,437,437,437,437,437,
2318: 437,437,437,437,437,437,437,437,437,437,437,437,437,437,437,
2319: 437,437,437,437,437,437,437,437,438,438,438,438,438,438,438,
2320: 438,438,438,438,438,438,438,438,438,438,438,438,438,438,438,
2321: 438,438,438,438,438,438,438,438,438,438,438,438,438,438,438,
2322: 439,439,439,439,439,439,439,439,439,439,439,439,439,439,439,
2323: 439,439,439,439,439,439,439,439,439,439,439,439,439,439,439,
2324: 439,439,439,439,439,439,439,439,439,439,439,439,439,439,439,
2325:
2326: /* block 130 */
2327: 440,440,440,440,440,440,440,440,440,440,440,440,440,440,440,
2328: 440,440,440,440,440,440,440,440,440,440,440,440,440, 87, 87,
2329: 441,441,441,441,441,441,441,441,441,441, 87, 87, 87, 87, 87, 87,
2330: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2331: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2332: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2333: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2334: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2335:

2336: /* block 131 */
2337: 442,442,442,442,442,442, 87, 87,442, 87,442,442,442,442,442,
2338: 442,442,442,442,442,442,442,442,442,442,442,442,442,442,
2339: 442,442,442,442,442,442,442,442,442,442,442,442,442,442,
2340: 442,442,442,442,442,442, 87,442,442, 87, 87, 87,442, 87, 87,442,
2341: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2342: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2343: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2344: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2345:
2346: /* block 132 */
2347: 443,443,443,443,443,443,443,443,443,443,443,443,443,443,
2348: 443,443,443,443,443,443,444,444,444,444, 87, 87, 87, 87, 87,445,
2349: 446,446,446,446,446,446,446,446,446,446,446,446,446,446,
2350: 446,446,446,446,446,446,446,446,446,446, 87, 87, 87, 87, 87,447,
2351: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2352: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2353: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2354: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2355:
2356: /* block 133 */
2357: 448,449,449,449, 87,449,449, 87, 87, 87, 87, 87,449,449,449,449,
2358: 448,448,448,448, 87,448,448,448, 87,448,448,448,448,448,448,
2359: 448,448,448,448,448,448,448,448,448,448,448,448,448,448,
2360: 448,448,448,448, 87, 87, 87, 87,449,449,449, 87, 87, 87, 87,449,
2361: 450,450,450,450,450,450,450,450, 87, 87, 87, 87, 87, 87, 87, 87,
2362: 451,451,451,451,451,451,451,451,451, 87, 87, 87, 87, 87, 87, 87,
2363: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2364: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2365:
2366: /* block 134 */
2367: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2368: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2369: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2370: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2371: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2372: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2373: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2374: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2375:
2376: /* block 135 */

2377: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2378: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2379: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2380: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2381: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2382: 452,452,452,452,452,452,452,452,452,452,452,452,452,452,
2383: 452,452,452,452,452,452,452,452,452,452,452,452,452, 87,
2384: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2385:
2386: /* block 136 */
2387: 453,453,453,453,453,453,453,453,453,453,453,453,453,453,
2388: 453,453,453,453,453,453,453,453,453,453,453,453,453,453,
2389: 453,453,453,453,453,453,453,453,453,453,453,453,453,453,
2390: 453,453,453,453,453,453,453,453,453,453,453,453,453,453,
2391: 453,453,453,453,453,453,453,453,453,453,453,453,453,453,
2392: 453,453,453,453,453,453,453,453,453,453,453,453,453,453,
2393: 453,453,453, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2394: 454,454,454,454, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2395:
2396: /* block 137 */
2397: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2398: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2399: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2400: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2401: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2402: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2403: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2404: 13, 13, 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2405:
2406: /* block 138 */
2407: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2408: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2409: 13, 13, 13, 13, 13, 13, 13, 87, 87, 13, 13, 13, 13, 13, 13,
2410: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2411: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2412: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2413: 13, 13, 13, 13, 13,455,455, 82, 82, 82, 13, 13, 13,455,455,455,
2414: 455,455,455, 16, 16, 16, 16, 16, 16, 16, 16, 82, 82, 82, 82,
2415:
2416: /* block 139 */
2417: 82, 82, 82, 13, 13, 82, 82, 82, 82, 82, 82, 13, 13, 13, 13,

2418: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2419: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 82, 82, 82, 82, 13, 13,
2420: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2421: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2422: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 87,
2423: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2424: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2425:
2426: /* block 140 */
2427: 425,425,425,425,425,425,425,425,425,425,425,425,425,425,425,
2428: 425,425,425,425,425,425,425,425,425,425,425,425,425,425,425,
2429: 425,425,425,425,425,425,425,425,425,425,425,425,425,425,425,
2430: 425,425,425,425,425,425,425,425,425,425,425,425,425,425,425,
2431: 425,425,456,456,456,425, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2432: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2433: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2434: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2435:
2436: /* block 141 */
2437: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2438: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2439: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2440: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2441: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2442: 13, 13, 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2443: 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
2444: 17, 17, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2445:
2446: /* block 142 */
2447: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2448: 343,343,343,343,343,343,343,343,343,343,344,344,344,344,344,
2449: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2450: 344,344,344,344,343,343,343,343,343,343,343,343,343,343,343,
2451: 343,343,343,343,343,343,343,343,343,343,343,343,343,344,344,
2452: 344,344,344,344,344, 87,344,344,344,344,344,344,344,344,344,
2453: 344,344,344,344,344,344,344,343,343,343,343,343,343,343,343,
2454: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2455:
2456: /* block 143 */
2457: 343,343,344,344,344,344,344,344,344,344,344,344,344,344,344,
2458: 344,344,344,344,344,344,344,344,344,344,343, 87,343,343,

2459: 87, 87,343, 87, 87,343,343, 87, 87,343,343,343,343, 87,343,343,
2460: 343,343,343,343,343,343,344,344,344,344, 87,344, 87,344,344,344,
2461: 344,344,344,344, 87,344,344,344,344,344,344,344,344,344,344,
2462: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2463: 343,343,343,343,343,343,343,343,343,343,344,344,344,344,344,344,
2464: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2465:
2466: /* block 144 */
2467: 344,344,344,344,343,343, 87,343,343,343,343, 87, 87,343,343,343,
2468: 343,343,343,343,343, 87,343,343,343,343,343,343,343, 87,344,344,
2469: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2470: 344,344,344,344,344,344,344,344,343,343, 87,343,343,343,343, 87,
2471: 343,343,343,343,343, 87,343, 87, 87, 87,343,343,343,343,343,343,
2472: 343, 87,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2473: 344,344,344,344,344,344,344,344,344,344,344,343,343,343,343,
2474: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2475:
2476: /* block 145 */
2477: 343,343,343,343,343,343,344,344,344,344,344,344,344,344,344,
2478: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2479: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2480: 343,343,343,343,343,343,343,343,343,343,344,344,344,344,344,
2481: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2482: 344,344,344,344,343,343,343,343,343,343,343,343,343,343,343,
2483: 343,343,343,343,343,343,343,343,343,343,343,343,343,344,344,
2484: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2485:
2486: /* block 146 */
2487: 344,344,344,344,344,344,344,344,343,343,343,343,343,343,343,
2488: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2489: 343,343,344,344,344,344,344,344,344,344,344,344,344,344,
2490: 344,344,344,344,344,344,344,344,344,344,344,343,343,343,343,
2491: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2492: 343,343,343,343,343,343,344,344,344,344,344,344,344,344,344,
2493: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2494: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2495:
2496: /* block 147 */
2497: 343,343,343,343,343,343,343,343,343,343,344,344,344,344,344,
2498: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2499: 344,344,344,344,344,344, 87, 87,343,343,343,343,343,343,343,343,

2500: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2501: 343, 6,344,344,344,344,344,344,344,344,344,344,344,344,344,
2502: 344,344,344,344,344,344,344,344,344,344, 6,344,344,344,344,
2503: 344,344,343,343,343,343,343,343,343,343,343,343,343,343,343,
2504: 343,343,343,343,343,343,343,343,343,343, 6,344,344,344,344,
2505:
2506: /* block 148 */
2507: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2508: 344,344,344,344,344, 6,344,344,344,344,344,344,343,343,343,343,
2509: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2510: 343,343,343,343,343, 6,344,344,344,344,344,344,344,344,344,344,
2511: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344, 6,
2512: 344,344,344,344,344,344,343,343,343,343,343,343,343,343,343,343,
2513: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343, 6,
2514: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2515:
2516: /* block 149 */
2517: 344,344,344,344,344,344,344,344,344, 6,344,344,344,344,344,344,
2518: 343,343,343,343,343,343,343,343,343,343,343,343,343,343,343,
2519: 343,343,343,343,343,343,343,343,343, 6,344,344,344,344,344,344,
2520: 344,344,344,344,344,344,344,344,344,344,344,344,344,344,344,
2521: 344,344,344, 6,344,344,344,344,344,344,343,344, 87, 87, 8, 8,
2522: 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
2523: 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
2524: 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
2525:
2526: /* block 150 */
2527: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2528: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2529: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 87, 87, 87, 87,
2530: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2531: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2532: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2533: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2534: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2535:
2536: /* block 151 */
2537: 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
2538: 13, 13, 13, 13, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2539: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2540: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,

2541: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2542: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2543: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2544: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2545:
2546: /* block 152 */
2547: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2548: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2549: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2550: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2551: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2552: 384,384,384,384,384,384,384, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2553: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2554: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2555:
2556: /* block 153 */
2557: 384,384,384,384,384,384,384,384,384,384,384,384,384,384,384,
2558: 384,384,384,384,384,384,384,384,384,384,384,384,384, 87, 87,
2559: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2560: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2561: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2562: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2563: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2564: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2565:
2566: /* block 154 */
2567: 87, 16, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2568: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2569: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
2570: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
2571: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
2572: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
2573: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
2574: 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
2575:
2576: /* block 155 */
2577: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2578: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2579: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2580: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2581: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,

```

2582: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2583: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2584: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2585:
2586: /* block 156 */
2587: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2588: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2589: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2590: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2591: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2592: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2593: 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82, 82,
2594: 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87, 87,
2595:
2596: /* block 157 */
2597: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2598: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2599: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2600: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2601: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2602: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2603: 420,420,420,420,420,420,420,420,420,420,420,420,420,420,420,
2604: 420,420,420,420,420,420,420,420,420,420,420,420,420, 87, 87,
2605:
2606: };
2607:
2608: #if UCD_BLOCK_SIZE != 128
2609: #error Please correct UCD_BLOCK_SIZE in pcre_internal.h
2610: #endif

```

File: sdm/VxWorks/libRegex/ucp.h

```
1: /*****
2: *      Unicode Property Table handler      *
3: *****/
4:
5: #ifndef _UCP_H
6: #define _UCP_H
7:
8: /* This file contains definitions of the property values that are returned by
9: the function _pcre_ucp_findprop(). New values that are added for new releases
10: of Unicode should always be at the end of each enum, for backwards
11: compatibility. */
12:
13: /* These are the general character categories. */
14:
15: enum {
16:  ucp_C,   /* Other */
17:  ucp_L,   /* Letter */
18:  ucp_M,   /* Mark */
19:  ucp_N,   /* Number */
20:  ucp_P,   /* Punctuation */
21:  ucp_S,   /* Symbol */
22:  ucp_Z    /* Separator */
23: };
24:
25: /* These are the particular character types. */
26:
27: enum {
28:  ucp_Cc,  /* Control */
29:  ucp_Cf,  /* Format */
30:  ucp_Cn,  /* Unassigned */
31:  ucp_Co,  /* Private use */
32:  ucp-Cs,  /* Surrogate */
33:  ucp_Ll,  /* Lower case letter */
34:  ucp_Lm,  /* Modifier letter */
35:  ucp_Lo,  /* Other letter */
36:  ucp_Lt,  /* Title case letter */
37:  ucp_Lu,  /* Upper case letter */
38:  ucp_Mc,  /* Spacing mark */
39:  ucp_Me,  /* Enclosing mark */
```

```

40: ucp_Mn, /* Non-spacing mark */
41: ucp_Nd, /* Decimal number */
42: ucp_Nl, /* Letter number */
43: ucp_No, /* Other number */
44: ucp_Pc, /* Connector punctuation */
45: ucp_Pd, /* Dash punctuation */
46: ucp_Pe, /* Close punctuation */
47: ucp_Pf, /* Final punctuation */
48: ucp_Pi, /* Initial punctuation */
49: ucp_Po, /* Other punctuation */
50: ucp_Ps, /* Open punctuation */
51: ucp_Sc, /* Currency symbol */
52: ucp_Sk, /* Modifier symbol */
53: ucp_Sm, /* Mathematical symbol */
54: ucp_So, /* Other symbol */
55: ucp_Zl, /* Line separator */
56: ucp_Zp, /* Paragraph separator */
57: ucp_Zs /* Space separator */
58: };
59:
60: /* These are the script identifications. */
61:
62: enum {
63: ucp_Arabic,
64: ucp_Armenian,
65: ucp_Bengali,
66: ucp_Bopomofo,
67: ucp_Braille,
68: ucp_Buginese,
69: ucp_Buhid,
70: ucp_Canadian_Aboriginal,
71: ucp_Cherokee,
72: ucp_Common,
73: ucp_Coptic,
74: ucp_Cypriot,
75: ucp_Cyrillic,
76: ucp_Deseret,
77: ucp_Devanagari,
78: ucp_Ethiopic,
79: ucp_Georgian,
80: ucp_Glagolitic,

```

81: ucp_Gothic,
82: ucp_Greek,
83: ucp_Gujarati,
84: ucp_Gurmukhi,
85: ucp_Han,
86: ucp_Hangul,
87: ucp_Hanunoo,
88: ucp_Hebrew,
89: ucp_Hiragana,
90: ucp_Inherited,
91: ucp_Kannada,
92: ucp_Katakana,
93: ucp_Kharoshthi,
94: ucp_Khmer,
95: ucp_Lao,
96: ucp_Latin,
97: ucp_Limbu,
98: ucp_Linear_B,
99: ucp_Malayalam,
100: ucp_Mongolian,
101: ucp_Myanmar,
102: ucp_New_Tai_Lue,
103: ucp_Ogham,
104: ucp_Old_Italic,
105: ucp_Old_Persian,
106: ucp_Oriya,
107: ucp_Osmanya,
108: ucp_Runic,
109: ucp_Shavian,
110: ucp_Sinhala,
111: ucp_Syloti_Nagri,
112: ucp_Syriac,
113: ucp_Tagalog,
114: ucp_Tagbanwa,
115: ucp_Tai_Le,
116: ucp_Tamil,
117: ucp_Telugu,
118: ucp_Thaana,
119: ucp_Thai,
120: ucp_Tibetan,
121: ucp_Tifinagh,

```
122: ucp_Ugaritic,
123: ucp_Yi,
124: /* New for Unicode 5.0: */
125: ucp_Balinese,
126: ucp_Cuneiform,
127: ucp_Nko,
128: ucp_Phags_Pa,
129: ucp_Phoenician,
130: /* New for Unicode 5.1: */
131: ucp_Carian,
132: ucp_Cham,
133: ucp_Kayah_Li,
134: ucp_Lepcha,
135: ucp_Lycian,
136: ucp_Lydian,
137: ucp_OL_Chiki,
138: ucp_Rejang,
139: ucp_Saurashtra,
140: ucp_Sundanese,
141: ucp_Vai
142: };
143:
144: #endif
145:
146: /* End of ucp.h */
```

File: sdm/VxWorks/apps/converter/converter.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMCancelxTEDS.h"
3: #include "../common/message/SDMmessage.h"
4: #include "../common/message/SDMSerreqst.h"
5: #include "../common/message/SDMData.h"
6: #include "../common/MessageManager/MessageManager.h"
7: #include "../common/message/SDMHello.h"
8: #include "../common/message/SDMID.h"
9: #include "../common/message/SDMRegister.h"
10: #include "../common/Time/SDMTime.h"
11: #include <string.h>
12: #include <stdio.h>
13:
14: void PerformRequest(char*);
15: void Register(MessageManager*);
16: double GetCurTime();
17: void CancelxTEDS();
18:
19: long my_port;
20: bool done = false;
21: int requestCount = 0;
22:
23: //xTEDS data
24: const char* XML_HEADER = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?>";
25: const char* XTEDS_SECTION = " \n<xTEDS version= \"2.0 \" name= \"converter_xTEDS \"/>";
26: const char* APP_SECTION = " \n \t<Application name= \"converter \" kind= \"Software \"/>";
27: const char* INTERFACE = " \n \t<Interface name= \"Converter_Interface \" id= \"1 \"/>";
28: const char* VAR_DATA_1 = " \n \t \t<Variable name= \"data \" format= \"FLOAT32 \"/>";
29: const char* VAR_DATA_2 = " \t \t \t<Variable name= \"data \" format= \"FLOAT32 \"/>";
30: const char* VAR_CONVERTEE_1 = " \n \t \t<Variable name= \"convertee \" format= \"UINT16 \"/>";
31: const char* VAR_CONVERTEE_2 = " \t \t \t<Variable name= \"convertee \" format= \"UINT16 \"/>";
32: const char* REQUEST = " \n \n \t \t<Request>";
33: const char* CMD_CONVERT_1 = " \n \t \t \t<CommandMsg name= \"convert \" id= \"1 \"/>";
34: const char* CMD_CONVERT_2 = " \n \t \t \t \t<VariableRef name= \"convertee \"/>";
35: const char* CMD_CONVERT_3 = " \n \t \t \t</CommandMsg>";
36: const char* MSG_RESULTS_1 = " \n \t \t \t<DataReplyMsg name= \"results \" id= \"2 \"/>";
37: const char* MSG_RESULTS_2 = " \t \t \t \t \t<DataReplyMsg name= \"results \" id= \"2 \"/>";
38: const char* MSG_RESULTS_3 = " \t \t \t \t \t<DataReplyMsg name= \"results \" id= \"2 \"/>";
39: const char* REQUEST_END = " \n \n \t \t</Request>";
```

```

40: const char* INTERFACE_END = "\n \t</Interface>";
41: const char* XTEDS_END = "\n</xTEDS>";
42:
43: int main(int argc,char** argv)
44: {
45:     SDMInit(argc,argv);
46:     my_port = getPort();
47:     if(my_port == SDM_PM_NOT_AVAILABLE)
48:     {
49:         printf("No PM is available to get port from! \n");
50:         printf("Using port 5003 \n");
51:         my_port = 5003;
52:     }
53:
54:     MessageManager mm;
55:     mm.Async_Init(my_port);
56:     Register(&mm);
57:
58:     char buf[BUFSIZE];
59:     // now wait for and handle 15 service request messages
60:     printf("Waiting for service request. \n");
61:     while(!done)
62:     {
63:         if(mm.IsReady())
64:         {
65:             switch(mm.GetMessage(buf))
66:             {
67:                 case SDM_Serreqst:
68:                     printf("Request Received \n");
69:                     requestCount++;
70:                     PerformRequest(buf);
71:                     break;
72:                 default:
73:                     break;
74:             }
75:         }
76:         if(requestCount >= 15)
77:         {
78:             done = true;
79:         }
80:         usleep(1000);

```



```

81: }
82: printf("Converter is shutting itself down \n");
83: CancelxTEDS();
84: return 0;
85: }
86:
87: void Register(MessageManager* mm)
88: {
89: bool helloReceived = false;
90: bool registerReceived = false;
91: bool idReceived = false;
92: bool registering = true;
93:
94: char buf[BUFSIZE];
95:
96: // create an xTEDS registration message
97: SDMxTEDS xteds;
98:
99: // SDMCancelxTEDS cancel;
100:
101: // set xTEDS
102: strcat (xteds.xTEDS,XML_HEADER);
103: strcat (xteds.xTEDS,XTEDS_SECTION);
104: strcat (xteds.xTEDS,APP_SECTION);
105: strcat (xteds.xTEDS,INTERFACE);
106: strcat (xteds.xTEDS,VAR_DATA_1);
107: strcat (xteds.xTEDS,VAR_DATA_2);
108: strcat (xteds.xTEDS,VAR_CONVERTEE_1);
109: strcat (xteds.xTEDS,VAR_CONVERTEE_2);
110: strcat (xteds.xTEDS,REQUEST);
111: strcat (xteds.xTEDS,CMD_CONVERT_1);
112: strcat (xteds.xTEDS,CMD_CONVERT_2);
113: strcat (xteds.xTEDS,CMD_CONVERT_3);
114: strcat (xteds.xTEDS,MSG_RESULTS_1);
115: strcat (xteds.xTEDS,MSG_RESULTS_2);
116: strcat (xteds.xTEDS,MSG_RESULTS_3);
117: strcat (xteds.xTEDS,REQUEST_END);
118: strcat (xteds.xTEDS,INTERFACE_END);
119: strcat (xteds.xTEDS,XTEDS_END);
120:
121: // set the id of this application

```

```

122:  xteds.source.setSensorID(1);
123:  //cancel.source.setSensorID(xteds.source.getSensorID());
124:  //cancel.source.setPort(my_port);
125:  xteds.source.setPort(my_port);
126:  printf("Registering converter xTEDS on port %ld \n",my_port);
127:  // register with the SDM
128:
129:  SDMHHello hello;
130:  hello.source.setPort(my_port);
131:  hello.type = 'A';
132:  double endTime = 0;
133:  while(registering)
134:  {
135:      if(mm->IsReady()) //Listen for messages
136:      {
137:          switch(mm->GetMessage(buf))
138:          {
139:              case SDM_ACK:
140:                  printf("SDMAck received acknowledging Hello \n");
141:                  helloReceived = true;
142:                  endTime = 0;
143:                  break;
144:              case SDM_Register:
145:                  printf("SDMRegister received \n");
146:                  registerReceived = true;
147:                  endTime = 0;
148:                  break;
149:              case SDM_ID:
150:                  printf("SDMID recieved, registration complete \n");
151:                  idReceived = true;
152:                  registering = false;
153:                  break;
154:              default:
155:                  break;
156:          }
157:      }
158:      else
159:      {
160:          if(GetCurTime() > endTime) //Check for timeouts
161:          {
162:              if(!helloReceived) //Send Hello Msg

```

```

163:         {
164:             printf("Sending SDMHHello \n");
165:             hello.Send();
166:             endTime = GetCurTime() + 5.0;
167:         }
168:         else if(registerReceived && !idReceived) //Register xTEDS
169:         {
170:             xteds.Send();
171:             endTime = GetCurTime() + 10.0;
172:         }
173:     }
174: }
175:     usleep(1000);
176: }
177: }
178:
179:
180: void PerformRequest(char* buf)
181: {
182:     unsigned short int_data;
183:     float float_data;
184:     SDMSerreqst request;
185:     SDMData dat;
186:
187:     request.Unmarshal(buf);
188:     // copy integer parameter into variable int_strain
189:     int_data = GET_SHORT(request.data);
190:     // convert to float between 0 and 1
191:     float_data = int_data/(float)0xffff;
192:     printf("Converted %d to %f \n",int_data,float_data);
193:     dat.source = request.source;
194:     // fill in other fields
195:     dat.msg_id = request.reply_id;
196:     // convert float_data to raw data bytes
197:     PUT_FLOAT(dat.msg, float_data);
198:     printf("Sending reply. \n");
199:     // return to requester
200:     dat.Send (request.destination,4);
201: }
202:
203: double GetCurTime()

```

```
204: {
205:     struct timespec tv;
206:     int tv_usec;
207:     clock_gettime(CLOCK_REALTIME, &tv);
208:     tv_usec = tv.tv_nsec/1000;
209:     return tv.tv_sec + ((double)tv_usec/1000000.0);
210: }
211:
212: void CancelxTEDS()
213: {
214:     SDMCancelxTEDS cancel;
215:     printf("Canceling xTEDS \n");
216:     cancel.source.setSensorID(1);
217:     cancel.source.setPort(my_port);
218:     cancel.Send();
219: }
```

File: sdm/VxWorks/apps/producer/producer.cpp

```
1: #include "../common/message/SDMxTEDS.h"
2: #include "../common/message/SDMSubreqst.h"
3: #include "../common/message/SDMDeletesub.h"
4: #include "../common/message/SDMCancelxTEDS.h"
5: #include "../common/SubscriptionManager/SubscriptionManager.h"
6: #include "../common/MessageManager/MessageManager.h"
7: #include "../common/message/SDMHeartbeat.h"
8: #include "../common/Time/SDMTime.h"
9: #include "../common/message/SDMHello.h"
10: #include "../common/message/SDMID.h"
11: #include "../common/message/SDMRegister.h"
12: #include <string.h>
13: #include <stdio.h>
14: #include <stdlib.h>
15: #include <pthread.h>
16:
17: const char* XML_HEADER = "<?xml version= \"1.0 \" encoding= \"UTF-8 \"?> \n";
18: const char* XTEDS_HEADER = "<xTEDS version= \"2.0 \" name= \"Producer_xTEDS \">";
19: const char* APP_SECTION = " \n \t<Application name= \"producer \" kind= \"data \"/>";
20: const char* INTERFACE_SECTION = " \n \t<Interface name= \"Producer_Interface \" id= \"1 \">";
21: const char* VAR_DATA_1 = " \n \t<Variable name= \"data \" format= \"UINT16 \" ";
22: const char* VAR_DATA_2 = "kind= \"data \"/>";
23: const char* NOTIFICATION = " \n \t<Notification>";
24: const char* MSG_ALL_1 = " \n \t \t<DataMsg name= \"all \" id= \"1 \" ";
25: const char* MSG_ALL_2 = "msgArrival= \"PERIODIC \" msgRate= \"1 \">";
26: const char* MSG_ALL_3 = " \n \t \t \t<VariableRef name= \"data \"/>";
27: const char* MSG_ALL_4 = " \n \t \t</DataMsg>";
28: const char* NOTIFICATION_END = " \n \t</Notification>";
29: const char* INTERFACE_END = " \n \t</Interface>";
30: const char* XTEDS_END = " \n</xTEDS> \n";
31:
32: void RegisterxTEDS();
33: void CancelxTEDS();
34: void* Publisher(void *);
35: void* Listener(void *);
36: double GetCurTime();
37:
38: SubscriptionManager subscriptions;
39: pthread_mutex_t subscription_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```

40: long myPort;
41: SDMID myID;
42: bool helloReply = false;
43: bool waitForReg = true;
44: bool waitForID = true;
45: const unsigned int THREAD_STACK_SIZE = 128000;
46:
47: int main(int argc,char** argv)
48: {
49: pthread_t ListenerThread;
50: pthread_t PublisherThread;
51:
52: SDMInit(argc,argv);
53: myPort = getPort();
54: if(myPort == SDM_PM_NOT_AVAILABLE)
55: {
56:     printf("No PM is available to get port from! \n");
57:     printf("Using port 5001 \n");
58:     myPort = 5001;
59: }
60:
61: pthread_attr_t threadAttr;
62: pthread_attr_init(&threadAttr);
63: pthread_attr_setstacksize(&threadAttr, THREAD_STACK_SIZE);
64:
65: pthread_create(&ListenerThread,&threadAttr,&Listener,NULL);
66: usleep(1000);
67: SDMHHello hello;
68: hello.source.setPort(myPort);
69: hello.type = 'A';
70: double endTime = 0;
71: double timeOut = 5.0;
72: while(!helloReply)
73: {
74:     if(GetCurTime() > endTime)
75:     {
76:         hello.Send();
77:         printf("Sending Hello \n");
78:         endTime = GetCurTime() + timeOut;
79:     }
80:     usleep(10000);

```

```

81: }
82: while (waitForReg)
83: {
84:     usleep(10000);
85: }
86: printf("Registering xTEDS \n");
87: RegisterxTEDS();
88: pthread_create(&PublisherThread,&threadAttr,&Publisher,NULL);
89: pthread_join(PublisherThread,NULL);
90: CancelxTEDS();
91: pthread_cancel(ListenerThread);
92: pthread_join(ListenerThread,NULL);
93: }
94:
95: void* Publisher(void * args)
96: {
97:     int published = 0;
98:     short data;
99:     while(published < 10)
100:     {
101:         data = (short)(rand()&0x00FF);
102:         char bufdata[2];
103:         PUT_SHORT(bufdata, data);
104:         pthread_mutex_lock(&subscription_mutex);
105:         if (subscriptions.Publish(1,1,bufdata,2))
106:         {
107:             published++;
108:         }
109:         pthread_mutex_unlock(&subscription_mutex);
110:         printf("Produced %d \tPublished %d / 10 \n",data,published);
111:         sleep(1);
112:     }
113:     return NULL;
114: }
115:
116: void* Listener(void * args)
117: {
118:     char buf[BUFSIZE];
119:     SDMSubreqst sub;
120:     SDMDeletesub del;
121:     MessageManager mm;

```

```

122:   mm.Async_Init(myPort);
123:   while(1)
124:   {
125:       pthread_testcancel();
126:       if(mm.IsReady())
127:       {
128:           SendHeartbeat();
129: #ifdef WIN32
130:           switch(mm.GetMsg(buf))
131: #else
132:           switch(mm.GetMessage(buf))
133: #endif
134:           {
135:               case SDM_ACK:
136:                   printf("SDMAck received \n");
137:                   helloReply = true;
138:                   break;
139:               case SDM_Register:
140:                   printf("SDMRegister received \n");
141:                   waitForReg = false;
142:                   break;
143:               case SDM_ID:
144:                   printf("SDMID received \n");
145:                   myID.Unmarshal(buf);
146:                   printf("CompID: \n");
147:                   printf("  SensorID: %li \n", myID.destination.getSensorID());
148:                   printf("  Address: %lx \n", myID.destination.getAddress());
149:                   printf("  Port: %i \n", myID.destination.getPort());
150:                   waitForID = false;
151:                   break;
152:               case SDM_Subreqst:
153:                   sub.Unmarshal(buf);
154:                   printf("Subscription Rec'd for %d \n", sub.msg_id.getInterfaceMessagePair());
155:                   pthread_mutex_lock(&subscription_mutex);
156:                   subscriptions.AddSubscription(sub);
157:                   pthread_mutex_unlock(&subscription_mutex);
158:                   break;
159:               case SDM_Deletesub:
160:                   printf("Cancel Rec'd \n");
161:                   del.Unmarshal(buf);
162:                   pthread_mutex_lock(&subscription_mutex);

```



```

163:         subscriptions.RemoveSubscription(del);
164:         pthread_mutex_unlock(&subscription_mutex);
165:         break;
166:     default:
167:         printf("Invalid Message found! \n");
168:         break;
169:     }
170: }
171: else
172: {
173:     usleep(100000);
174: }
175: }
176: return NULL;
177: }
178:
179: void RegisterxTEDS()
180: {
181:     // create an xTEDS registration message
182:     SDMxTEDS xteds;
183:
184:     // set xTEDS
185:     strcat (xteds.xTEDS,XML_HEADER);
186:     strcat (xteds.xTEDS,XTEDS_HEADER);
187:     strcat (xteds.xTEDS,APP_SECTION);
188:     strcat (xteds.xTEDS,INTERFACE_SECTION);
189:     strcat (xteds.xTEDS,VAR_DATA_1);
190:     strcat (xteds.xTEDS,VAR_DATA_2);
191:     strcat (xteds.xTEDS,NOTIFICATION);
192:     strcat (xteds.xTEDS,MSG_ALL_1);
193:     strcat (xteds.xTEDS,MSG_ALL_2);
194:     strcat (xteds.xTEDS,MSG_ALL_3);
195:     strcat (xteds.xTEDS,MSG_ALL_4);
196:     strcat (xteds.xTEDS,NOTIFICATION_END);
197:     strcat (xteds.xTEDS,INTERFACE_END);
198:     strcat (xteds.xTEDS,XTEDS_END);
199:
200:     // set the id of this application
201:     xteds.source.setSensorID(1);
202:     xteds.source.setPort(myPort);
203:     printf("Registering producer xTEDS on port %ld \n",myPort);

```

```

204: // register with the SDM
205: xteds.Send();
206: }
207:
208: void CancelxTEDS()
209: {
210:     SDMCancelxTEDS cancel;
211:     printf("Canceling xTEDS \n");
212:     cancel.source.setSensorID(1);
213:     cancel.source.setPort(myPort);
214:     cancel.Send();
215: }
216:
217: double GetCurTime()
218: {
219:     struct timespec tv;
220:     int tv_usec;
221:     clock_gettime(CLOCK_REALTIME, &tv);
222:     tv_usec = tv.tv_nsec/1000;
223:     return tv.tv_sec + ((double)tv_usec/1000000.0);
224: }

```

File: sdm/VxWorks/apps/consumer/consumer.cpp

```
1: #include "../common/message/SDMData.h"
2: #include "../common/message/SDMService.h"
3: #include "../common/message/SDMConsume.h"
4: #include "../common/message/SDMRegInfo.h"
5: #include "../common/message/SDMReqReg.h"
6: #include "../common/message/SDMHeartbeat.h"
7: #include "../common/MessageManipulator/MessageManipulator.h"
8: #include "../common/MessageManager/MessageManager.h"
9: #include "../common/Time/SDMTime.h"
10:
11: #include <string.h>
12: #include <stdio.h>
13:
14: #define DATA_PROVIDER    1
15: #define SERVICE_PROVIDER 2
16:
17: long my_port;
18: SDMComponent_ID data_provider;
19: SDMComponent_ID service_provider;
20: SDMMessage_ID data_msg(0,0);
21: SDMMessage_ID service_msg(0,0);
22:
23: MessageManipulator data_manipulator;
24: MessageManipulator service_manipulator;
25:
26: void DataHandler(SDMData& dat,long length)
27: {
28:     SDMService request;
29:     short int_value;
30:     float float_value;
31:     static float geo_average = 0;
32:
33:     if((dat.source == data_provider)&&(dat.msg_id == data_msg))
34:     {
35:         //marshal appropriate service message
36:         //copy source componentID into service request
37:         request.source=service_provider;
38:         request.destination.setPort(my_port);
39:         //copy msg id into service request
```

```

40:     request.command_id=service_msg;
41:     //set the length we are sending
42:     request.length = sizeof(short);
43:     //copy integer data into service request
44:     if(service_provider.getSensorID()==0)
45:     {
46:         printf("No converter is available \n");
47:         return;
48:     }
49:     int_value = data_manipulator.getUINT16Value("data",dat,DATAMSG);
50:     service_manipulator.setValue("convertee",request,int_value);
51:     //send request
52:     request.Send();
53: }
54: else if((dat.source == service_provider))
55: {
56:     float_value = service_manipulator.getFloat32Value("data",dat,DATAMSG);
57:     geo_average /=2;
58:     geo_average += float_value;
59:     //extract and display results
60:     printf("Running Average -- %f \n",geo_average);
61: }
62: }
63:
64: void RegInfoHandler(SDMRegInfo& info)
65: {
66:     SDMConsume consume;
67:     SDMReqReg req_reg;
68:
69:     //Set the port we will be receiving on
70:     consume.destination.setPort(my_port);
71:     //copy the sensor id into the consume message
72:     consume.source=info.source;
73:     //copy the msg id into the consume message
74:     consume.msg_id=info.msg_id;
75:
76:     switch(info.id)
77:     {
78:     case DATA_PROVIDER:
79:         if(info.type == 1)
80:         {

```

```

81:     data_provider.setSensorID(0);
82:     printf("Data provider failed . . . ");
83:     printf("Searching for new provider \n");
84:     //request info on integer data providers,
85:     //the DM will repost software tasks that
86:     // could satisfiy our requirments
87:
88:     //Set variable name
89:     strcpy(req_reg.item_name,"data");
90:     //Set the quallist can be empty
91:     strcpy(req_reg.quallist,"< format= \"UINT16 \"/>");
92:     req_reg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
93:     req_reg.destination.setPort(my_port);
94:     req_reg.id = DATA_PROVIDER;
95:     req_reg.Send();
96: }
97: else
98: {
99:     if(data_provider.getSensorID() == 0)
100:     {
101:         data_provider = info.source;
102:         data_msg = info.msg_id;
103:         data_manipulator.setMsgDef(info.msg_def);
104:         printf("New Data provider found");
105:         printf(" \n \n%s \n",info.msg_def);
106:         //Send the consume message
107:         consume.Send();
108:     }
109:     else
110:     {
111:         printf("Data provider found");
112:         printf(" - not used \n \n%s \n",info.msg_def);
113:     }
114: }
115: break;
116: case SERVICE_PROVIDER:
117:     if(info.type == SDM_REGINFO_CANCELLATION)
118:     {
119:         service_provider.setSensorID(0);
120:         printf("Service provider failed . . . ");
121:         printf("Searching for new provider \n");

```

```

122:         //request info on service providers,
123:         //the DM will repost software tasks
124:         //that could satisfy our requirements
125:
126:         //Set var name to convert which we already know
127:         strcpy(req_reg.item_name,"convert");
128:         //We will not be using wildcards
129:         req_reg.quallist[0] = '\0';
130:         req_reg.reply = SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
131:         req_reg.destination.setPort(my_port);
132:         req_reg.id = SERVICE_PROVIDER;
133:         req_reg.Send();
134:     }
135:     else
136:     {
137:         if(service_provider.getSensorID() == 0)
138:         {
139:             service_provider = info.source;
140:             service_msg = info.msg_id;
141:             service_manipulator.setMsgDef(info.msg_def);
142:             printf("New Service provider found");
143:             printf("\n\n%s\n",info.msg_def);
144:         }
145:         else
146:         {
147:             printf("Service provider found - ");
148:             printf("\n\n%s\n",info.msg_def);
149:         }
150:     }
151:     break;
152: }
153:
154: }
155:
156: int main(int argc,char** argv)
157: {
158:     MessageManager mm;
159:     SDMDData dat;
160:     SDMRegInfo info;
161:     SDMReqReg req_reg;
162:     char buf[BUFSIZE];

```

```

163: long length;
164:
165: //initialize consumer
166: SDMInit(argc,argv);
167: my_port = getPort();
168: if(my_port == SDM_PM_NOT_AVAILABLE)
169: {
170:     printf("No PM is available to get port from! \n");
171:     printf("Using port 5002 \n");
172:     my_port = 5002;
173: }
174: mm.Async_Init(my_port);
175:
176: printf("Consumer listening on port %ld \n",my_port);
177:
178: while(1)
179: {
180:     if (mm.IsReady())
181:     {
182:         SendHeartbeat();
183: #ifdef WIN32
184:         switch(mm.GetMsg(buf,length))
185: #else
186:         switch(mm.GetMessage(buf,length))
187: #endif
188:         {
189:             case SDM_Data:
190:                 dat.Unmarshal(buf,length);
191:                 DataHandler(dat,length);
192:                 break;
193:             case SDM_RegInfo:
194:                 if(info.Unmarshal(buf)!=SDM_NO_FURTHER_DATA_PROVIDER)
195:                 {
196:                     RegInfoHandler(info);
197:                 }
198:                 break;
199:             default:
200:                 printf("Unexpected message \n");
201:         }
202:     }
203:     else

```

```

204:    { //check for data and service providers
205:        if(data_provider.getSensorID() == 0)
206:        {
207:            //request info on integer data providers
208:            //Set variable name
209:            strcpy(req_reg.item_name,"data");
210:            //Set the quallist can be empty
211:            strcpy(req_reg.quallist,"< format= \"UINT16 \"/>");
212:            req_reg.reply                                     =
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
213:            req_reg.destination.setPort(my_port);
214:            req_reg.id = DATA_PROVIDER;
215:            req_reg.Send();
216:            printf("Searching for new data provider \n");
217:            sleep(2);
218:        }
219:        if(service_provider.getSensorID() == 0)
220:        {
221:            //request info on service providers
222:            //Set var name to convert which we already know
223:            strcpy(req_reg.item_name,"convert");
224:            //We will not be using wildcards
225:            req_reg.quallist[0] = '\0';
226:            req_reg.reply                                     =
SDM_REQREG_CURRENT_FUTURE_AND_CANCELLATIONS;
227:            req_reg.destination.setPort(my_port);
228:            req_reg.id = SERVICE_PROVIDER;
229:            req_reg.Send();
230:            printf("Searching for new service provider \n");
231:            sleep(2);
232:        }
233:        usleep(1000);
234:    }
235: }
236: }

```


Listing from directory: sdm/Win32

File: sdm/Win32/unix/semaphore.h

```
1: /*
2:  * Module: semaphore.h
3:  *
4:  * Purpose:
5:  *   Semaphores aren't actually part of the PThreads standard.
6:  *   They are defined by the POSIX Standard:
7:  *
8:  *       POSIX 1003.1b-1993   (POSIX.1b)
9:  *
10:  * -----
11:  *
12:  *   Pthreads-win32 - POSIX Threads Library for Win32
13:  *   Copyright(C) 1998 John E. Bossom
14:  *   Copyright(C) 1999,2005 Pthreads-win32 contributors
15:  *
16:  *   Contact Email: rpj@callisto.canberra.edu.au
17:  *
18:  *   The current list of contributors is contained
19:  *   in the file CONTRIBUTORS included with the source
20:  *   code distribution. The list can also be seen at the
21:  *   following World Wide Web location:
22:  *   http://sources.redhat.com/pthreads-win32/contributors.html
23:  *
24:  *   This library is free software; you can redistribute it and/or
25:  *   modify it under the terms of the GNU Lesser General Public
26:  *   License as published by the Free Software Foundation; either
27:  *   version 2 of the License, or (at your option) any later version.
28:  *
29:  *   This library is distributed in the hope that it will be useful,
30:  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
31:  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
32:  *   Lesser General Public License for more details.
33:  *
34:  *   You should have received a copy of the GNU Lesser General Public
35:  *   License along with this library in the file COPYING.LIB;
36:  *   if not, write to the Free Software Foundation, Inc.,
37:  *   59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
```

```

38: */
39: #if !defined( SEMAPHORE_H )
40: #define SEMAPHORE_H
41:
42: #undef PTW32_LEVEL
43:
44: #if defined(_POSIX_SOURCE)
45: #define PTW32_LEVEL 0
46: /* Early POSIX */
47: #endif
48:
49: #if defined(_POSIX_C_SOURCE) && _POSIX_C_SOURCE >= 199309
50: #undef PTW32_LEVEL
51: #define PTW32_LEVEL 1
52: /* Include 1b, 1c and 1d */
53: #endif
54:
55: #if defined(INCLUDE_NP)
56: #undef PTW32_LEVEL
57: #define PTW32_LEVEL 2
58: /* Include Non-Portable extensions */
59: #endif
60:
61: #define PTW32_LEVEL_MAX 3
62:
63: #if !defined(PTW32_LEVEL)
64: #define PTW32_LEVEL PTW32_LEVEL_MAX
65: /* Include everything */
66: #endif
67:
68: #if __GNUC__ && ! defined (__declspec)
69: # error Please upgrade your GNU compiler to one that supports __declspec.
70: #endif
71:
72: /*
73:  * When building the DLL code, you should define PTW32_BUILD so that
74:  * the variables/functions are exported correctly. When using the DLL,
75:  * do NOT define PTW32_BUILD, and then the variables/functions will
76:  * be imported correctly.
77:  */
78: #ifndef PTW32_STATIC_LIB

```

```

79: # ifdef PTW32_BUILD
80: #   define PTW32_DLLPORT __declspec (dllexport)
81: # else
82: #   define PTW32_DLLPORT __declspec (dllimport)
83: # endif
84: #else
85: # define PTW32_DLLPORT
86: #endif
87:
88: /*
89:  * This is a duplicate of what is in the autoconf config.h,
90:  * which is only used when building the pthread-win32 libraries.
91:  */
92:
93: #ifndef PTW32_CONFIG_H
94: # if defined(WINCE)
95: #   define NEED_ERRNO
96: #   define NEED_SEM
97: # endif
98: # if defined(_UWIN) || defined(__MINGW32__)
99: #   define HAVE_MODE_T
100: # endif
101: #endif
102:
103: /*
104:  *
105:  */
106:
107: #if PTW32_LEVEL >= PTW32_LEVEL_MAX
108: #ifndef NEED_ERRNO
109: #include "need_errno.h"
110: #else
111: #include <errno.h>
112: #endif
113: #endif /* PTW32_LEVEL >= PTW32_LEVEL_MAX */
114:
115: #define _POSIX_SEMAPHORES
116:
117: #ifdef __cplusplus
118: extern "C"
119: {

```

```

120: #endif          /* __cplusplus */
121:
122: #ifndef HAVE_MODE_T
123: typedef unsigned int mode_t;
124: #endif
125:
126:
127: typedef struct sem_t * sem_t;
128:
129: PTW32_DLLPORT int __cdecl sem_init (sem_t * sem,
130:          int pshared,
131:          unsigned int value);
132:
133: PTW32_DLLPORT int __cdecl sem_destroy (sem_t * sem);
134:
135: PTW32_DLLPORT int __cdecl sem_trywait (sem_t * sem);
136:
137: PTW32_DLLPORT int __cdecl sem_wait (sem_t * sem);
138:
139: PTW32_DLLPORT int __cdecl sem_timedwait (sem_t * sem,
140:          const struct timespec * abstime);
141:
142: PTW32_DLLPORT int __cdecl sem_post (sem_t * sem);
143:
144: PTW32_DLLPORT int __cdecl sem_post_multiple (sem_t * sem,
145:          int count);
146:
147: PTW32_DLLPORT int __cdecl sem_open (const char * name,
148:          int oflag,
149:          mode_t mode,
150:          unsigned int value);
151:
152: PTW32_DLLPORT int __cdecl sem_close (sem_t * sem);
153:
154: PTW32_DLLPORT int __cdecl sem_unlink (const char * name);
155:
156: PTW32_DLLPORT int __cdecl sem_getvalue (sem_t * sem,
157:          int * sval);
158:
159: #ifdef __cplusplus
160: }          /* End of extern "C" */

```

```
161: #endif          /* __cplusplus */
162:
163: #undef PTW32_LEVEL
164: #undef PTW32_LEVEL_MAX
165:
166: #endif          /* !SEMAPHORE_H */
```

File: sdm/Win32/unix/poll.cpp

```
1: #include "sys/poll.h"
2: #include <stdio.h>
3: #include <unistd.h>
4: #define WIN32_LEAN_AND_MEAN
5: #include <winsock2.h>
6: #include "sdmLib.h"
7:
8:
9: SDMLIB_API int poll(struct pollfd* fds, unsigned int nfds, int timeout)
10: {
11:     int num = 0;
12:     u_long len;
13:
14:     for( unsigned int i=0; i<nfds; i++ )
15:     {
16:         if ( fds[i].events & (POLLIN|POLLPRI) )
17:         {
18:             len = 0;
19:             ioctlsocket(fds[i].fd,FIONREAD, &len );
20:             if ( len > 0 )
21:                 num++;
22:         }
23:     }
24:     if ( num == 0 && timeout != 0 && timeout != INFINITE )
25:         usleep(timeout*1000);
26:     else
27:         return num;
28:     for( unsigned int i=0; i<nfds; i++ )
29:     {
30:         if ( fds[i].events & (POLLIN|POLLPRI) )
31:         {
32:             len = 0;
33:             ioctlsocket(fds[i].fd,FIONREAD, &len );
34:             if ( len > 0 )
35:                 num++;
36:         }
37:     }
38:     return num;
39: }
```

File: sdm/Win32/unix/endian.h

```
1: #ifndef __endian_h_
2: #define __endian_h_
3:
4: #ifdef _WIN32
5: #define __BYTE_ORDER __LITTLE_ENDIAN
6: #endif
7:
8: #endif // __endian_h_
9:
```

File: sdm/Win32/unix/sched.h

```
1: /*
2:  * Module: sched.h
3:  *
4:  * Purpose:
5:  *   Provides an implementation of POSIX realtime extensions
6:  *   as defined in
7:  *
8:  *       POSIX 1003.1b-1993    (POSIX.1b)
9:  *
10:  * -----
11:  *
12:  *   Pthreads-win32 - POSIX Threads Library for Win32
13:  *   Copyright(C) 1998 John E. Bossom
14:  *   Copyright(C) 1999,2005 Pthreads-win32 contributors
15:  *
16:  *   Contact Email: rpj@callisto.canberra.edu.au
17:  *
18:  *   The current list of contributors is contained
19:  *   in the file CONTRIBUTORS included with the source
20:  *   code distribution. The list can also be seen at the
21:  *   following World Wide Web location:
22:  *   http://sources.redhat.com/pthreads-win32/contributors.html
23:  *
24:  *   This library is free software; you can redistribute it and/or
25:  *   modify it under the terms of the GNU Lesser General Public
26:  *   License as published by the Free Software Foundation; either
27:  *   version 2 of the License, or (at your option) any later version.
28:  *
29:  *   This library is distributed in the hope that it will be useful,
30:  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
31:  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
32:  *   Lesser General Public License for more details.
33:  *
34:  *   You should have received a copy of the GNU Lesser General Public
35:  *   License along with this library in the file COPYING.LIB;
36:  *   if not, write to the Free Software Foundation, Inc.,
37:  *   59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
38:  */
39: #ifndef _SCHED_H
```



```

40: #define _SCHED_H
41:
42: #undef PTW32_LEVEL
43:
44: #if defined(_POSIX_SOURCE)
45: #define PTW32_LEVEL 0
46: /* Early POSIX */
47: #endif
48:
49: #if defined(_POSIX_C_SOURCE) && _POSIX_C_SOURCE >= 199309
50: #undef PTW32_LEVEL
51: #define PTW32_LEVEL 1
52: /* Include 1b, 1c and 1d */
53: #endif
54:
55: #if defined(INCLUDE_NP)
56: #undef PTW32_LEVEL
57: #define PTW32_LEVEL 2
58: /* Include Non-Portable extensions */
59: #endif
60:
61: #define PTW32_LEVEL_MAX 3
62:
63: #if !defined(PTW32_LEVEL)
64: #define PTW32_LEVEL PTW32_LEVEL_MAX
65: /* Include everything */
66: #endif
67:
68:
69: #if __GNUC__ && !defined(__declspec)
70: # error Please upgrade your GNU compiler to one that supports __declspec.
71: #endif
72:
73: /*
74:  * When building the DLL code, you should define PTW32_BUILD so that
75:  * the variables/functions are exported correctly. When using the DLL,
76:  * do NOT define PTW32_BUILD, and then the variables/functions will
77:  * be imported correctly.
78:  */
79: #ifndef PTW32_STATIC_LIB
80: # ifdef PTW32_BUILD

```

```

81: # define PTW32_DLLPORT __declspec (dllexport)
82: # else
83: # define PTW32_DLLPORT __declspec (dllimport)
84: # endif
85: #else
86: # define PTW32_DLLPORT
87: #endif
88:
89: /*
90:  * This is a duplicate of what is in the autoconf config.h,
91:  * which is only used when building the pthread-win32 libraries.
92:  */
93:
94: #ifndef PTW32_CONFIG_H
95: # if defined(WINCE)
96: #   define NEED_ERRNO
97: #   define NEED_SEM
98: # endif
99: # if defined(_UWIN) || defined(__MINGW32__)
100: #   define HAVE_MODE_T
101: # endif
102: #endif
103:
104: /*
105:  *
106:  */
107:
108: #if PTW32_LEVEL >= PTW32_LEVEL_MAX
109: #ifdef NEED_ERRNO
110: #include "need_errno.h"
111: #else
112: #include <errno.h>
113: #endif
114: #endif /* PTW32_LEVEL >= PTW32_LEVEL_MAX */
115:
116: #if defined(__MINGW32__) || defined(_UWIN)
117: #if PTW32_LEVEL >= PTW32_LEVEL_MAX
118: /* For pid_t */
119: # include <sys/types.h>
120: /* Required by Unix 98 */
121: # include <time.h>

```

```

122: #endif /* PTW32_LEVEL >= PTW32_LEVEL_MAX */
123: #else
124: // #if !defined (WIN32)
125: typedef int pid_t;
126: // #endif
127: #endif
128:
129: /* Thread scheduling policies */
130:
131: enum {
132:  SCHED_OTHER = 0,
133:  SCHED_FIFO,
134:  SCHED_RR,
135:  SCHED_MIN  = SCHED_OTHER,
136:  SCHED_MAX  = SCHED_RR
137: };
138:
139: struct sched_param {
140:  int sched_priority;
141: };
142:
143: #ifdef __cplusplus
144: extern "C"
145: {
146: #endif          /* __cplusplus */
147:
148: PTW32_DLLPORT int __cdecl sched_yield (void);
149:
150: PTW32_DLLPORT int __cdecl sched_get_priority_min (int policy);
151:
152: PTW32_DLLPORT int __cdecl sched_get_priority_max (int policy);
153:
154: PTW32_DLLPORT int __cdecl sched_setscheduler (pid_t pid, int policy);
155:
156: PTW32_DLLPORT int __cdecl sched_getscheduler (pid_t pid);
157:
158: /*
159:  * Note that this macro returns ENOTSUP rather than
160:  * ENOSYS as might be expected. However, returning ENOSYS
161:  * should mean that sched_get_priority_{min,max} are
162:  * not implemented as well as sched_rr_get_interval.

```

```

163: * This is not the case, since we just don't support
164: * round-robin scheduling. Therefore I have chosen to
165: * return the same value as sched_setscheduler when
166: * SCHED_RR is passed to it.
167: */
168: #define sched_rr_get_interval(_pid, _interval) \
169: ( errno = ENOTSUP, (int) -1 )
170:
171:
172: #ifdef __cplusplus
173: }                /* End of extern "C" */
174: #endif          /* __cplusplus */
175:
176: #undef PTW32_LEVEL
177: #undef PTW32_LEVEL_MAX
178:
179: #endif          /* !_SCHED_H */
180:

```

File: sdm/Win32/unix/socket.cpp

```
1: #include <sys/socket.h>
2: #include <stdio.h>
3: #include "sdmLib.h"
4:
5: int setsockopt( int socket, unsigned int level, unsigned int option, void* value, size_t valueSize )
6: {
7:     return setsockopt(socket, level, option, (const char*)value, (int)valueSize );
8: }
```

File: sdm/Win32/unix/pthread.h

```
1: /* This is an implementation of the threads API of POSIX 1003.1-2001.
2: *
3: * -----
4: *
5: *   Pthreads-win32 - POSIX Threads Library for Win32
6: *   Copyright(C) 1998 John E. Bossom
7: *   Copyright(C) 1999,2005 Pthreads-win32 contributors
8: *
9: *   Contact Email: rpj@callisto.canberra.edu.au
10: *
11: *   The current list of contributors is contained
12: *   in the file CONTRIBUTORS included with the source
13: *   code distribution. The list can also be seen at the
14: *   following World Wide Web location:
15: *   http://sources.redhat.com/pthreads-win32/contributors.html
16: *
17: *   This library is free software; you can redistribute it and/or
18: *   modify it under the terms of the GNU Lesser General Public
19: *   License as published by the Free Software Foundation; either
20: *   version 2 of the License, or (at your option) any later version.
21: *
22: *   This library is distributed in the hope that it will be useful,
23: *   but WITHOUT ANY WARRANTY; without even the implied warranty of
24: *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
25: *   Lesser General Public License for more details.
26: *
27: *   You should have received a copy of the GNU Lesser General Public
28: *   License along with this library in the file COPYING.LIB;
29: *   if not, write to the Free Software Foundation, Inc.,
30: *   59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
31: */
32:
33: #if !defined( PTHREAD_H )
34: #define PTHREAD_H
35:
36: /*
37: * See the README file for an explanation of the pthreads-win32 version
38: * numbering scheme and how the DLL is named etc.
39: */
```

```

40: #define PTW32_VERSION 2,7,0,0
41: #define PTW32_VERSION_STRING "2, 7, 0, 0 \0"
42:
43: /* There are three implementations of cancel cleanup.
44:  * Note that pthread.h is included in both application
45:  * compilation units and also internally for the library.
46:  * The code here and within the library aims to work
47:  * for all reasonable combinations of environments.
48:  *
49:  * The three implementations are:
50:  *
51:  * WIN32 SEH
52:  * C
53:  * C++
54:  *
55:  * Please note that exiting a push/pop block via
56:  * "return", "exit", "break", or "continue" will
57:  * lead to different behaviour amongst applications
58:  * depending upon whether the library was built
59:  * using SEH, C++, or C. For example, a library built
60:  * with SEH will call the cleanup routine, while both
61:  * C++ and C built versions will not.
62:  */
63:
64: /*
65:  * Define defaults for cleanup code.
66:  * Note: Unless the build explicitly defines one of the following, then
67:  * we default to standard C style cleanup. This style uses setjmp/longjmp
68:  * in the cancelation and thread exit implementations and therefore won't
69:  * do stack unwinding if linked to applications that have it (e.g.
70:  * C++ apps). This is currently consistent with most/all commercial Unix
71:  * POSIX threads implementations.
72:  */
73: #if !defined( __CLEANUP_SEH ) && !defined( __CLEANUP_CXX ) && !defined(
__CLEANUP_C )
74: # define __CLEANUP_C
75: #endif
76:
77: #if defined( __CLEANUP_SEH ) && ( !defined( _MSC_VER ) && !defined(PTW32_RC_MSC))
78: #error ERROR [__FILE__, line __LINE__]: SEH is not supported for this compiler.
79: #endif

```

```

80:
81: /*
82:  * Stop here if we are being included by the resource compiler.
83:  */
84: #ifndef RC_INVOKED
85:
86: #undef PTW32_LEVEL
87:
88: #if defined(_POSIX_SOURCE)
89: #define PTW32_LEVEL 0
90: /* Early POSIX */
91: #endif
92:
93: #if defined(_POSIX_C_SOURCE) && _POSIX_C_SOURCE >= 199309
94: #undef PTW32_LEVEL
95: #define PTW32_LEVEL 1
96: /* Include 1b, 1c and 1d */
97: #endif
98:
99: #if defined(INCLUDE_NP)
100: #undef PTW32_LEVEL
101: #define PTW32_LEVEL 2
102: /* Include Non-Portable extensions */
103: #endif
104:
105: #define PTW32_LEVEL_MAX 3
106:
107: #if !defined(PTW32_LEVEL)
108: #define PTW32_LEVEL PTW32_LEVEL_MAX
109: /* Include everything */
110: #endif
111:
112: #ifdef _UWIN
113: # define HAVE_STRUCT_TIMESPEC 1
114: # define HAVE_SIGNAL_H      1
115: # undef HAVE_CONFIG_H
116: # pragma comment(lib, "pthread")
117: #endif
118:
119: /*
120:  * -----

```


121: *
122: *
123: * Module: pthread.h
124: *
125: * Purpose:
126: * Provides an implementation of PThreads based upon the
127: * standard:
128: *
129: * POSIX 1003.1-2001
130: * and
131: * The Single Unix Specification version 3
132: *
133: * (these two are equivalent)
134: *
135: * in order to enhance code portability between Windows,
136: * various commercial Unix implementations, and Linux.
137: *
138: * See the ANNOUNCE file for a full list of conforming
139: * routines and defined constants, and a list of missing
140: * routines and constants not defined in this implementation.
141: *
142: * Authors:
143: * There have been many contributors to this library.
144: * The initial implementation was contributed by
145: * John Bosson, and several others have provided major
146: * sections or revisions of parts of the implementation.
147: * Often significant effort has been contributed to
148: * find and fix important bugs and other problems to
149: * improve the reliability of the library, which sometimes
150: * is not reflected in the amount of code which changed as
151: * result.
152: * As much as possible, the contributors are acknowledged
153: * in the ChangeLog file in the source code distribution
154: * where their changes are noted in detail.
155: *
156: * Contributors are listed in the CONTRIBUTORS file.
157: *
158: * As usual, all bouquets go to the contributors, and all
159: * brickbats go to the project maintainer.
160: *
161: * Maintainer:

```

162: *   The code base for this project is coordinated and
163: *   eventually pre-tested, packaged, and made available by
164: *
165: *       Ross Johnson <rpj@callisto.canberra.edu.au>
166: *
167: * QA Testers:
168: *   Ultimately, the library is tested in the real world by
169: *   a host of competent and demanding scientists and
170: *   engineers who report bugs and/or provide solutions
171: *   which are then fixed or incorporated into subsequent
172: *   versions of the library. Each time a bug is fixed, a
173: *   test case is written to prove the fix and ensure
174: *   that later changes to the code don't reintroduce the
175: *   same error. The number of test cases is slowly growing
176: *   and therefore so is the code reliability.
177: *
178: * Compliance:
179: *   See the file ANNOUNCE for the list of implemented
180: *   and not-implemented routines and defined options.
181: *   Of course, these are all defined in this file as well.
182: *
183: * Web site:
184: *   The source code and other information about this library
185: *   are available from
186: *
187: *       http://sources.redhat.com/pthreads-win32/
188: *
189: * -----
190: */
191:
192: /* Try to avoid including windows.h */
193: #if defined(__MINGW32__) && defined(__cplusplus)
194: #define PTW32_INCLUDE_WINDOWS_H
195: #endif
196:
197: #ifndef PTW32_INCLUDE_WINDOWS_H
198: #include <windows.h>
199: #endif
200:
201: #if defined(_MSC_VER) && _MSC_VER < 1300 || defined(__DMC__)
202: /*

```

```

203: * VC++6.0 or early compiler's header has no DWORD_PTR type.
204: */
205: typedef unsigned long DWORD_PTR;
206: #endif
207: /*
208: * -----
209: * autoconf switches
210: * -----
211: */
212:
213: #if HAVE_CONFIG_H
214: #include "config.h"
215: #endif /* HAVE_CONFIG_H */
216:
217: #ifndef NEED_FTIME
218: #include <time.h>
219: #else /* NEED_FTIME */
220: /* use native WIN32 time API */
221: #endif /* NEED_FTIME */
222:
223: #if HAVE_SIGNAL_H
224: #include <signal.h>
225: #endif /* HAVE_SIGNAL_H */
226:
227: #include <setjmp.h>
228: #include <limits.h>
229:
230: /*
231: * Boolean values to make us independent of system includes.
232: */
233: enum {
234:   PTW32_FALSE = 0,
235:   PTW32_TRUE = (! PTW32_FALSE)
236: };
237:
238: /*
239: * This is a duplicate of what is in the autoconf config.h,
240: * which is only used when building the pthread-win32 libraries.
241: */
242:
243: #ifndef PTW32_CONFIG_H

```

```

244: # if defined(WINCE)
245: #   define NEED_ERRNO
246: #   define NEED_SEM
247: # endif
248: # if defined(_UWIN) || defined(__MINGW32__)
249: #   define HAVE_MODE_T
250: # endif
251: #endif
252:
253: /*
254: *
255: */
256:
257: #if PTW32_LEVEL >= PTW32_LEVEL_MAX
258: #ifdef NEED_ERRNO
259: #include "need_errno.h"
260: #else
261: #include <errno.h>
262: #endif
263: #endif /* PTW32_LEVEL >= PTW32_LEVEL_MAX */
264:
265: /*
266: * Several systems don't define some error numbers.
267: */
268: #ifndef ENOTSUP
269: #   define ENOTSUP 48 /* This is the value in Solaris. */
270: #endif
271:
272: #ifndef ETIMEDOUT
273: #   define ETIMEDOUT 10060 /* This is the value in winsock.h. */
274: #endif
275:
276: #ifndef ENOSYS
277: #   define ENOSYS 140 /* Semi-arbitrary value */
278: #endif
279:
280: #ifndef EDEADLK
281: #   ifdef EDEADLOCK
282: #       define EDEADLK EDEADLOCK
283: #   else
284: #       define EDEADLK 36 /* This is the value in MSVC. */

```

```

285: # endif
286: #endif
287:
288: #include <sched.h>
289:
290: /*
291:  * To avoid including windows.h we define only those things that we
292:  * actually need from it.
293:  */
294: #ifndef PTW32_INCLUDE_WINDOWS_H
295: #ifndef HANDLE
296: # define PTW32__HANDLE_DEF
297: # define HANDLE void *
298: #endif
299: #ifndef DWORD
300: # define PTW32__DWORD_DEF
301: # define DWORD unsigned long
302: #endif
303: #endif
304:
305: #ifndef HAVE_STRUCT_TIMESPEC
306: #define HAVE_STRUCT_TIMESPEC 1
307: struct timespec {
308:     long tv_sec;
309:     long tv_nsec;
310: };
311: #endif /* HAVE_STRUCT_TIMESPEC */
312:
313: #ifndef SIG_BLOCK
314: #define SIG_BLOCK 0
315: #endif /* SIG_BLOCK */
316:
317: #ifndef SIG_UNBLOCK
318: #define SIG_UNBLOCK 1
319: #endif /* SIG_UNBLOCK */
320:
321: #ifndef SIG_SETMASK
322: #define SIG_SETMASK 2
323: #endif /* SIG_SETMASK */
324:
325: #ifdef __cplusplus

```

```

326: extern "C"
327: {
328: #endif          /* __cplusplus */
329:
330: /*
331: * -----
332: *
333: * POSIX 1003.1-2001 Options
334: * =====
335: *
336: * Options are normally set in <unistd.h>, which is not provided
337: * with pthreads-win32.
338: *
339: * For conformance with the Single Unix Specification (version 3), all of the
340: * options below are defined, and have a value of either -1 (not supported)
341: * or 200112L (supported).
342: *
343: * These options can neither be left undefined nor have a value of 0, because
344: * either indicates that sysconf(), which is not implemented, may be used at
345: * runtime to check the status of the option.
346: *
347: * _POSIX_THREADS (== 200112L)
348: *          If == 200112L, you can use threads
349: *
350: * _POSIX_THREAD_ATTR_STACKSIZE (== 200112L)
351: *          If == 200112L, you can control the size of a thread's
352: *          stack
353: *          pthread_attr_getstacksize
354: *          pthread_attr_setstacksize
355: *
356: * _POSIX_THREAD_ATTR_STACKADDR (== -1)
357: *          If == 200112L, you can allocate and control a thread's
358: *          stack. If not supported, the following functions
359: *          will return ENOSYS, indicating they are not
360: *          supported:
361: *          pthread_attr_getstackaddr
362: *          pthread_attr_setstackaddr
363: *
364: * _POSIX_THREAD_PRIORITY_SCHEDULING (== -1)
365: *          If == 200112L, you can use realtime scheduling.
366: *          This option indicates that the behaviour of some

```

367: * implemented functions conforms to the additional TPS
 368: * requirements in the standard. E.g. rwlocks favour
 369: * writers over readers when threads have equal priority.
 370: *
 371: * `_POSIX_THREAD_PRIO_INHERIT` (== -1)
 372: * If == 200112L, you can create priority inheritance
 373: * mutexes.
 374: * `pthread_mutexattr_getprotocol` +
 375: * `pthread_mutexattr_setprotocol` +
 376: *
 377: * `_POSIX_THREAD_PRIO_PROTECT` (== -1)
 378: * If == 200112L, you can create priority ceiling mutexes
 379: * Indicates the availability of:
 380: * `pthread_mutex_getprioceiling`
 381: * `pthread_mutex_setprioceiling`
 382: * `pthread_mutexattr_getprioceiling`
 383: * `pthread_mutexattr_getprotocol` +
 384: * `pthread_mutexattr_setprioceiling`
 385: * `pthread_mutexattr_setprotocol` +
 386: *
 387: * `_POSIX_THREAD_PROCESS_SHARED` (== -1)
 388: * If set, you can create mutexes and condition
 389: * variables that can be shared with another
 390: * process. If set, indicates the availability
 391: * of:
 392: * `pthread_mutexattr_getpshared`
 393: * `pthread_mutexattr_setpshared`
 394: * `pthread_condattr_getpshared`
 395: * `pthread_condattr_setpshared`
 396: *
 397: * `_POSIX_THREAD_SAFE_FUNCTIONS` (== 200112L)
 398: * If == 200112L you can use the special `*_r` library
 399: * functions that provide thread-safe behaviour
 400: *
 401: * `_POSIX_READER_WRITER_LOCKS` (== 200112L)
 402: * If == 200112L, you can use read/write locks
 403: *
 404: * `_POSIX_SPIN_LOCKS` (== 200112L)
 405: * If == 200112L, you can use spin locks
 406: *
 407: * `_POSIX_BARRIERS` (== 200112L)

```

408: *           If == 200112L, you can use barriers
409: *
410: *   + These functions provide both 'inherit' and/or
411: *     'protect' protocol, based upon these macro
412: *     settings.
413: *
414: * -----
415: */
416:
417: /*
418: * POSIX Options
419: */
420: #undef _POSIX_THREADS
421: #define _POSIX_THREADS 200112L
422:
423: #undef _POSIX_READER_WRITER_LOCKS
424: #define _POSIX_READER_WRITER_LOCKS 200112L
425:
426: #undef _POSIX_SPIN_LOCKS
427: #define _POSIX_SPIN_LOCKS 200112L
428:
429: #undef _POSIX_BARRIERS
430: #define _POSIX_BARRIERS 200112L
431:
432: #undef _POSIX_THREAD_SAFE_FUNCTIONS
433: #define _POSIX_THREAD_SAFE_FUNCTIONS 200112L
434:
435: #undef _POSIX_THREAD_ATTR_STACKSIZE
436: #define _POSIX_THREAD_ATTR_STACKSIZE 200112L
437:
438: /*
439: * The following options are not supported
440: */
441: #undef _POSIX_THREAD_ATTR_STACKADDR
442: #define _POSIX_THREAD_ATTR_STACKADDR -1
443:
444: #undef _POSIX_THREAD_PRIO_INHERIT
445: #define _POSIX_THREAD_PRIO_INHERIT -1
446:
447: #undef _POSIX_THREAD_PRIO_PROTECT
448: #define _POSIX_THREAD_PRIO_PROTECT -1

```



```

449:
450: /* TPS is not fully supported. */
451: #undef _POSIX_THREAD_PRIORITY_SCHEDULING
452: #define _POSIX_THREAD_PRIORITY_SCHEDULING -1
453:
454: #undef _POSIX_THREAD_PROCESS_SHARED
455: #define _POSIX_THREAD_PROCESS_SHARED -1
456:
457:
458: /*
459:  * POSIX 1003.1-2001 Limits
460:  * =====
461:  *
462:  * These limits are normally set in <limits.h>, which is not provided with
463:  * pthreads-win32.
464:  *
465:  * PTHREAD_DESTRUCTOR_ITERATIONS
466:  *      Maximum number of attempts to destroy
467:  *      a thread's thread-specific data on
468:  *      termination (must be at least 4)
469:  *
470:  * PTHREAD_KEYS_MAX
471:  *      Maximum number of thread-specific data keys
472:  *      available per process (must be at least 128)
473:  *
474:  * PTHREAD_STACK_MIN
475:  *      Minimum supported stack size for a thread
476:  *
477:  * PTHREAD_THREADS_MAX
478:  *      Maximum number of threads supported per
479:  *      process (must be at least 64).
480:  *
481:  * SEM_NSEMS_MAX
482:  *      The maximum number of semaphores a process can have.
483:  *      (must be at least 256)
484:  *
485:  * SEM_VALUE_MAX
486:  *      The maximum value a semaphore can have.
487:  *      (must be at least 32767)
488:  *
489:  */

```

```

490: #undef _POSIX_THREAD_DESTRUCTOR_ITERATIONS
491: #define _POSIX_THREAD_DESTRUCTOR_ITERATIONS    4
492:
493: #undef PTHREAD_DESTRUCTOR_ITERATIONS
494:         #define                                PTHREAD_DESTRUCTOR_ITERATIONS
_POSIX_THREAD_DESTRUCTOR_ITERATIONS
495:
496: #undef _POSIX_THREAD_KEYS_MAX
497: #define _POSIX_THREAD_KEYS_MAX                128
498:
499: #undef PTHREAD_KEYS_MAX
500: #define PTHREAD_KEYS_MAX                      _POSIX_THREAD_KEYS_MAX
501:
502: #undef PTHREAD_STACK_MIN
503: #define PTHREAD_STACK_MIN                    0
504:
505: #undef _POSIX_THREAD_THREADS_MAX
506: #define _POSIX_THREAD_THREADS_MAX            64
507:
508: /* Arbitrary value */
509: #undef PTHREAD_THREADS_MAX
510: #define PTHREAD_THREADS_MAX                  2019
511:
512: #undef _POSIX_SEM_NSEMS_MAX
513: #define _POSIX_SEM_NSEMS_MAX                256
514:
515: /* Arbitrary value */
516: #undef SEM_NSEMS_MAX
517: #define SEM_NSEMS_MAX                        1024
518:
519: #undef _POSIX_SEM_VALUE_MAX
520: #define _POSIX_SEM_VALUE_MAX                32767
521:
522: #undef SEM_VALUE_MAX
523: #define SEM_VALUE_MAX                        INT_MAX
524:
525:
526: #if __GNUC__ && ! defined (__declspec)
527: # error Please upgrade your GNU compiler to one that supports __declspec.
528: #endif
529:

```

```

530: /*
531:  * When building the DLL code, you should define PTW32_BUILD so that
532:  * the variables/functions are exported correctly. When using the DLL,
533:  * do NOT define PTW32_BUILD, and then the variables/functions will
534:  * be imported correctly.
535:  */
536: #ifndef PTW32_STATIC_LIB
537: #   ifdef PTW32_BUILD
538: #       define PTW32_DLLPORT __declspec (dllexport)
539: #   else
540: #       define PTW32_DLLPORT __declspec (dllimport)
541: #   endif
542: #else
543: #   define PTW32_DLLPORT
544: #endif
545:
546: /*
547:  * The Open Watcom C/C++ compiler uses a non-standard calling convention
548:  * that passes function args in registers unless __cdecl is explicitly specified
549:  * in exposed function prototypes.
550:  *
551:  * We force all calls to cdecl even though this could slow Watcom code down
552:  * slightly. If you know that the Watcom compiler will be used to build both
553:  * the DLL and application, then you can probably define this as a null string.
554:  * Remember that pthread.h (this file) is used for both the DLL and application builds.
555:  */
556: #define PTW32_CDECL __cdecl
557:
558: #if defined(_UWIN) && PTW32_LEVEL >= PTW32_LEVEL_MAX
559: #   include <sys/types.h>
560: #else
561: /*
562:  * Generic handle type - intended to extend uniqueness beyond
563:  * that available with a simple pointer. It should scale for either
564:  * IA-32 or IA-64.
565:  */
566: typedef struct {
567:     void * p;           /* Pointer to actual object */
568:     unsigned int x;      /* Extra information - reuse count etc */
569: } ptw32_handle_t;
570:

```

```

571: typedef ptw32_handle_t pthread_t;
572: typedef struct pthread_attr_t * pthread_attr_t;
573: typedef struct pthread_once_t pthread_once_t;
574: typedef struct pthread_key_t * pthread_key_t;
575: typedef struct pthread_mutex_t * pthread_mutex_t;
576: typedef struct pthread_mutexattr_t * pthread_mutexattr_t;
577: typedef struct pthread_cond_t * pthread_cond_t;
578: typedef struct pthread_condattr_t * pthread_condattr_t;
579: #endif
580: typedef struct pthread_rwlock_t * pthread_rwlock_t;
581: typedef struct pthread_rwlockattr_t * pthread_rwlockattr_t;
582: typedef struct pthread_spinlock_t * pthread_spinlock_t;
583: typedef struct pthread_barrier_t * pthread_barrier_t;
584: typedef struct pthread_barrierattr_t * pthread_barrierattr_t;
585:
586: /*
587:  * =====
588:  * =====
589:  * POSIX Threads
590:  * =====
591:  * =====
592:  */
593:
594: enum {
595: /*
596:  * pthread_attr_{get,set}detachstate
597:  */
598: PTHREAD_CREATE_JOINABLE    = 0, /* Default */
599: PTHREAD_CREATE_DETACHED    = 1,
600:
601: /*
602:  * pthread_attr_{get,set}inheritsched
603:  */
604: PTHREAD_INHERIT_SCHED      = 0,
605: PTHREAD_EXPLICIT_SCHED     = 1, /* Default */
606:
607: /*
608:  * pthread_{get,set}scope
609:  */
610: PTHREAD_SCOPE_PROCESS      = 0,
611: PTHREAD_SCOPE_SYSTEM       = 1, /* Default */

```

```

612:
613: /*
614:  * pthread_setcancelstate paramters
615:  */
616: PTHREAD_CANCEL_ENABLE      = 0, /* Default */
617: PTHREAD_CANCEL_DISABLE    = 1,
618:
619: /*
620:  * pthread_setcanceltype parameters
621:  */
622: PTHREAD_CANCEL_ASYNCHRONOUS = 0,
623: PTHREAD_CANCEL_DEFERRED     = 1, /* Default */
624:
625: /*
626:  * pthread_mutexattr_{get,set}pshared
627:  * pthread_condattr_{get,set}pshared
628:  */
629: PTHREAD_PROCESS_PRIVATE     = 0,
630: PTHREAD_PROCESS_SHARED      = 1,
631:
632: /*
633:  * pthread_barrier_wait
634:  */
635: PTHREAD_BARRIER_SERIAL_THREAD = -1
636: };
637:
638: /*
639:  * =====
640:  * =====
641:  * Cancellation
642:  * =====
643:  * =====
644:  */
645: #define PTHREAD_CANCELED      ((void *) -1)
646:
647:
648: /*
649:  * =====
650:  * =====
651:  * Once Key
652:  * =====

```

```

653: * =====
654: */
655: #define PTHREAD_ONCE_INIT    { PTW32_FALSE, 0, 0, 0}
656:
657: struct pthread_once_t
658: {
659:     int     done;    /* indicates if user function has been executed */
660:     void *   lock;
661:     int     reserved1;
662:     int     reserved2;
663: };
664:
665:
666: /*
667: * =====
668: * =====
669: * Object initialisers
670: * =====
671: * =====
672: */
673: #define PTHREAD_MUTEX_INITIALIZER ((pthread_mutex_t) -1)
674: #define PTHREAD_RECURSIVE_MUTEX_INITIALIZER ((pthread_mutex_t) -2)
675: #define PTHREAD_ERRORCHECK_MUTEX_INITIALIZER ((pthread_mutex_t) -3)
676:
677: /*
678: * Compatibility with LinuxThreads
679: */
680:         #define                PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP
PTHREAD_RECURSIVE_MUTEX_INITIALIZER
681:         #define                PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP
PTHREAD_ERRORCHECK_MUTEX_INITIALIZER
682:
683: #define PTHREAD_COND_INITIALIZER ((pthread_cond_t) -1)
684:
685: #define PTHREAD_RWLOCK_INITIALIZER ((pthread_rwlock_t) -1)
686:
687: #define PTHREAD_SPINLOCK_INITIALIZER ((pthread_spinlock_t) -1)
688:
689:
690: /*
691: * Mutex types.

```

```

692: */
693: enum
694: {
695:     /* Compatibility with LinuxThreads */
696:     PTHREAD_MUTEX_FAST_NP,
697:     PTHREAD_MUTEX_RECURSIVE_NP,
698:     PTHREAD_MUTEX_ERRORCHECK_NP,
699:     PTHREAD_MUTEX_TIMED_NP = PTHREAD_MUTEX_FAST_NP,
700:     PTHREAD_MUTEX_ADAPTIVE_NP = PTHREAD_MUTEX_FAST_NP,
701:     /* For compatibility with POSIX */
702:     PTHREAD_MUTEX_NORMAL = PTHREAD_MUTEX_FAST_NP,
703:     PTHREAD_MUTEX_RECURSIVE = PTHREAD_MUTEX_RECURSIVE_NP,
704:     PTHREAD_MUTEX_ERRORCHECK = PTHREAD_MUTEX_ERRORCHECK_NP,
705:     PTHREAD_MUTEX_DEFAULT = PTHREAD_MUTEX_NORMAL
706: };
707:
708:
709: typedef struct ptw32_cleanup_t ptw32_cleanup_t;
710:
711: #if defined(_MSC_VER)
712:     /* Disable MSVC 'anachronism used' warning */
713:     #pragma warning( disable : 4229 )
714: #endif
715:
716: typedef void (* PTW32_CDECL ptw32_cleanup_callback_t)(void *);
717:
718: #if defined(_MSC_VER)
719:     #pragma warning( default : 4229 )
720: #endif
721:
722: struct ptw32_cleanup_t
723: {
724:     ptw32_cleanup_callback_t routine;
725:     void *arg;
726:     struct ptw32_cleanup_t *prev;
727: };
728:
729: #ifdef __CLEANUP_SEH
730:     /*
731:      * WIN32 SEH version of cancel cleanup.
732:      */

```

```

733:
734: #define pthread_cleanup_push( _rout, _arg ) \
735:     { \
736:         ptw32_cleanup_t  _cleanup; \
737:         \
738:         _cleanup.routine  = (ptw32_cleanup_callback_t)(_rout); \
739:         _cleanup.arg      = (_arg); \
740:         __try \
741:         { \
742:
743: #define pthread_cleanup_pop( _execute ) \
744:         } \
745:         __finally \
746:         { \
747:             if( _execute || AbnormalTermination()) \
748:             { \
749:                 (*( _cleanup.routine))( _cleanup.arg ); \
750:             } \
751:         } \
752:     }
753:
754: #else /* __CLEANUP_SEH */
755:
756: #ifdef __CLEANUP_C
757:
758:     /*
759:      * C implementation of PThreads cancel cleanup
760:      */
761:
762: #define pthread_cleanup_push( _rout, _arg ) \
763:     { \
764:         ptw32_cleanup_t  _cleanup; \
765:         \
766:         ptw32_push_cleanup( &_cleanup, (ptw32_cleanup_callback_t) (_rout), (_arg) ); \
767:
768: #define pthread_cleanup_pop( _execute ) \
769:         (void) ptw32_pop_cleanup( _execute ); \
770:     }
771:
772: #else /* __CLEANUP_C */
773:

```



```

774: #ifdef __CLEANUP_CXX
775:
776:     /*
777:      * C++ version of cancel cleanup.
778:      * - John E. Bossom.
779:      */
780:
781:     class PThreadCleanup {
782:     /*
783:      * PThreadCleanup
784:      *
785:      * Purpose
786:      *   This class is a C++ helper class that is
787:      *   used to implement pthread_cleanup_push/
788:      *   pthread_cleanup_pop.
789:      *   The destructor of this class automatically
790:      *   pops the pushed cleanup routine regardless
791:      *   of how the code exits the scope
792:      *   (i.e. such as by an exception)
793:      */
794:     ptw32_cleanup_callback_t cleanUpRout;
795:     void *    obj;
796:     int      executeIt;
797:
798:     public:
799:     PThreadCleanup() :
800:         cleanUpRout( 0 ),
801:         obj( 0 ),
802:         executeIt( 0 )
803:     /*
804:      * No cleanup performed
805:      */
806:     {
807:     }
808:
809:     PThreadCleanup(
810:         ptw32_cleanup_callback_t routine,
811:         void *    arg ) :
812:         cleanUpRout( routine ),
813:         obj( arg ),
814:         executeIt( 1 )

```

```

815:      /*
816:       * Registers a cleanup routine for 'arg'
817:       */
818:      {
819:      }
820:
821: ~PThreadCleanup()
822: {
823:     if ( executeIt && ((void *) cleanUpRout != (void *) 0) )
824:     {
825:         (void) (*cleanUpRout)( obj );
826:     }
827: }
828:
829: void execute( int exec )
830: {
831:     executeIt = exec;
832: }
833: };
834:
835: /*
836:  * C++ implementation of PThreads cancel cleanup;
837:  * This implementation takes advantage of a helper
838:  * class who's destructor automatically calls the
839:  * cleanup routine if we exit our scope weirdly
840:  */
841: #define pthread_cleanup_push( _rout, _arg ) \
842: { \
843:     PThreadCleanup cleanup((ptw32_cleanup_callback_t)(_rout), \
844:         (void *) (_arg) );
845:
846: #define pthread_cleanup_pop( _execute ) \
847:     cleanup.execute( _execute ); \
848: }
849:
850: #else
851:
852: #error ERROR [__FILE__, line __LINE__]: Cleanup type undefined.
853:
854: #endif /* __CLEANUP_CXX */
855:

```

```

856: #endif /* __CLEANUP_C */
857:
858: #endif /* __CLEANUP_SEH */
859:
860: /*
861:  * =====
862:  * =====
863:  * Methods
864:  * =====
865:  * =====
866:  */
867:
868: /*
869:  * PThread Attribute Functions
870:  */
871: PTW32_DLLPORT int PTW32_CDECL pthread_attr_init (pthread_attr_t * attr);
872:
873: PTW32_DLLPORT int PTW32_CDECL pthread_attr_destroy (pthread_attr_t * attr);
874:
875: PTW32_DLLPORT int PTW32_CDECL pthread_attr_getdetachstate (const pthread_attr_t * attr,
876:                                                             int *detachstate);
877:
878: PTW32_DLLPORT int PTW32_CDECL pthread_attr_getstackaddr (const pthread_attr_t * attr,
879:                                                            void **stackaddr);
880:
881: PTW32_DLLPORT int PTW32_CDECL pthread_attr_getstacksize (const pthread_attr_t * attr,
882:                                                            size_t * stacksize);
883:
884: PTW32_DLLPORT int PTW32_CDECL pthread_attr_setdetachstate (pthread_attr_t * attr,
885:                                                             int detachstate);
886:
887: PTW32_DLLPORT int PTW32_CDECL pthread_attr_setstackaddr (pthread_attr_t * attr,
888:                                                            void *stackaddr);
889:
890: PTW32_DLLPORT int PTW32_CDECL pthread_attr_setstacksize (pthread_attr_t * attr,
891:                                                            size_t stacksize);
892:
893: PTW32_DLLPORT int PTW32_CDECL pthread_attr_getschedparam (const pthread_attr_t *attr,
894:                                                             struct sched_param *param);
895:
896: PTW32_DLLPORT int PTW32_CDECL pthread_attr_setschedparam (pthread_attr_t *attr,

```

```

897:                const struct sched_param *param);
898:
899: PTW32_DLLPORT int PTW32_CDECL pthread_attr_setschedpolicy (pthread_attr_t *,
900:                int);
901:
902: PTW32_DLLPORT int PTW32_CDECL pthread_attr_getschedpolicy (pthread_attr_t *,
903:                int *);
904:
905: PTW32_DLLPORT int PTW32_CDECL pthread_attr_setinheritsched(pthread_attr_t * attr,
906:                int inheritsched);
907:
908: PTW32_DLLPORT int PTW32_CDECL pthread_attr_getinheritsched(pthread_attr_t * attr,
909:                int * inheritsched);
910:
911: PTW32_DLLPORT int PTW32_CDECL pthread_attr_setscope (pthread_attr_t *,
912:                int);
913:
914: PTW32_DLLPORT int PTW32_CDECL pthread_attr_getscope (const pthread_attr_t *,
915:                int *);
916:
917: /*
918:  * PThread Functions
919:  */
920: PTW32_DLLPORT int PTW32_CDECL pthread_create (pthread_t * tid,
921:                const pthread_attr_t * attr,
922:                void *(*start) (void *),
923:                void *arg);
924:
925: PTW32_DLLPORT int PTW32_CDECL pthread_detach (pthread_t tid);
926:
927: PTW32_DLLPORT int PTW32_CDECL pthread_equal (pthread_t t1,
928:                pthread_t t2);
929:
930: PTW32_DLLPORT void PTW32_CDECL pthread_exit (void *value_ptr);
931:
932: PTW32_DLLPORT int PTW32_CDECL pthread_join (pthread_t thread,
933:                void **value_ptr);
934:
935: PTW32_DLLPORT pthread_t PTW32_CDECL pthread_self (void);
936:
937: PTW32_DLLPORT int PTW32_CDECL pthread_cancel (pthread_t thread);

```

```

938:
939: PTW32_DLLPORT int PTW32_CDECL pthread_setcancelstate (int state,
940:                int *oldstate);
941:
942: PTW32_DLLPORT int PTW32_CDECL pthread_setcanceltype (int type,
943:                int *oldtype);
944:
945: PTW32_DLLPORT void PTW32_CDECL pthread_testcancel (void);
946:
947: PTW32_DLLPORT int PTW32_CDECL pthread_once (pthread_once_t * once_control,
948:                void (*init_routine) (void));
949:
950: #if PTW32_LEVEL >= PTW32_LEVEL_MAX
951: PTW32_DLLPORT ptw32_cleanup_t * PTW32_CDECL ptw32_pop_cleanup (int execute);
952:
953: PTW32_DLLPORT void PTW32_CDECL ptw32_push_cleanup (ptw32_cleanup_t * cleanup,
954:                void (*routine) (void *),
955:                void *arg);
956: #endif /* PTW32_LEVEL >= PTW32_LEVEL_MAX */
957:
958: /*
959:  * Thread Specific Data Functions
960:  */
961: PTW32_DLLPORT int PTW32_CDECL pthread_key_create (pthread_key_t * key,
962:                void (*destructor) (void *));
963:
964: PTW32_DLLPORT int PTW32_CDECL pthread_key_delete (pthread_key_t key);
965:
966: PTW32_DLLPORT int PTW32_CDECL pthread_setspecific (pthread_key_t key,
967:                const void *value);
968:
969: PTW32_DLLPORT void * PTW32_CDECL pthread_getspecific (pthread_key_t key);
970:
971:
972: /*
973:  * Mutex Attribute Functions
974:  */
975: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_init (pthread_mutexattr_t * attr);
976:
977: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_destroy (pthread_mutexattr_t * attr);
978:

```

```

979: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_getpshared (const pthread_mutexattr_t
980:                      * attr,
981:                      int *pshared);
982:
983: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_setpshared (pthread_mutexattr_t * attr,
984:                      int pshared);
985:
986: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_settype (pthread_mutexattr_t * attr, int
987: kind);
988: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_gettype (pthread_mutexattr_t * attr, int
989: *kind);
990: /*
991:  * Barrier Attribute Functions
992:  */
993: PTW32_DLLPORT int PTW32_CDECL pthread_barrierattr_init (pthread_barrierattr_t * attr);
994: PTW32_DLLPORT int PTW32_CDECL pthread_barrierattr_destroy (pthread_barrierattr_t * attr);
995:
996: PTW32_DLLPORT int PTW32_CDECL pthread_barrierattr_getpshared (const pthread_barrierattr_t
997:                      * attr,
998:                      int *pshared);
999:
1000: PTW32_DLLPORT int PTW32_CDECL pthread_barrierattr_setpshared (pthread_barrierattr_t *
1001: attr,
1002:                      int pshared);
1003: /*
1004:  * Mutex Functions
1005:  */
1006: PTW32_DLLPORT int PTW32_CDECL pthread_mutex_init (pthread_mutex_t * mutex,
1007:                      const pthread_mutexattr_t * attr);
1008:
1009: PTW32_DLLPORT int PTW32_CDECL pthread_mutex_destroy (pthread_mutex_t * mutex);
1010:
1011: PTW32_DLLPORT int PTW32_CDECL pthread_mutex_lock (pthread_mutex_t * mutex);
1012:
1013: PTW32_DLLPORT int PTW32_CDECL pthread_mutex_timedlock(pthread_mutex_t *mutex,
1014:                      const struct timespec *abstime);
1015:
1016: PTW32_DLLPORT int PTW32_CDECL pthread_mutex_trylock (pthread_mutex_t * mutex);
1017:

```

```

1018: PTW32_DLLPORT int PTW32_CDECL pthread_mutex_unlock (pthread_mutex_t * mutex);
1019:
1020: /*
1021:  * Spinlock Functions
1022:  */
1023: PTW32_DLLPORT int PTW32_CDECL pthread_spin_init (pthread_spinlock_t * lock, int
pshared);
1024:
1025: PTW32_DLLPORT int PTW32_CDECL pthread_spin_destroy (pthread_spinlock_t * lock);
1026:
1027: PTW32_DLLPORT int PTW32_CDECL pthread_spin_lock (pthread_spinlock_t * lock);
1028:
1029: PTW32_DLLPORT int PTW32_CDECL pthread_spin_trylock (pthread_spinlock_t * lock);
1030:
1031: PTW32_DLLPORT int PTW32_CDECL pthread_spin_unlock (pthread_spinlock_t * lock);
1032:
1033: /*
1034:  * Barrier Functions
1035:  */
1036: PTW32_DLLPORT int PTW32_CDECL pthread_barrier_init (pthread_barrier_t * barrier,
1037:             const pthread_barrierattr_t * attr,
1038:             unsigned int count);
1039:
1040: PTW32_DLLPORT int PTW32_CDECL pthread_barrier_destroy (pthread_barrier_t * barrier);
1041:
1042: PTW32_DLLPORT int PTW32_CDECL pthread_barrier_wait (pthread_barrier_t * barrier);
1043:
1044: /*
1045:  * Condition Variable Attribute Functions
1046:  */
1047: PTW32_DLLPORT int PTW32_CDECL pthread_condattr_init (pthread_condattr_t * attr);
1048:
1049: PTW32_DLLPORT int PTW32_CDECL pthread_condattr_destroy (pthread_condattr_t * attr);
1050:
1051: PTW32_DLLPORT int PTW32_CDECL pthread_condattr_getpshared (const pthread_condattr_t *
attr,
1052:             int *pshared);
1053:
1054: PTW32_DLLPORT int PTW32_CDECL pthread_condattr_setpshared (pthread_condattr_t * attr,
1055:             int pshared);
1056:

```

```

1057: /*
1058:  * Condition Variable Functions
1059:  */
1060: PTW32_DLLPORT int PTW32_CDECL pthread_cond_init (pthread_cond_t * cond,
1061:          const pthread_condattr_t * attr);
1062:
1063: PTW32_DLLPORT int PTW32_CDECL pthread_cond_destroy (pthread_cond_t * cond);
1064:
1065: PTW32_DLLPORT int PTW32_CDECL pthread_cond_wait (pthread_cond_t * cond,
1066:          pthread_mutex_t * mutex);
1067:
1068: PTW32_DLLPORT int PTW32_CDECL pthread_cond_timedwait (pthread_cond_t * cond,
1069:          pthread_mutex_t * mutex,
1070:          const struct timespec *abstime);
1071:
1072: PTW32_DLLPORT int PTW32_CDECL pthread_cond_signal (pthread_cond_t * cond);
1073:
1074: PTW32_DLLPORT int PTW32_CDECL pthread_cond_broadcast (pthread_cond_t * cond);
1075:
1076: /*
1077:  * Scheduling
1078:  */
1079: PTW32_DLLPORT int PTW32_CDECL pthread_setschedparam (pthread_t thread,
1080:          int policy,
1081:          const struct sched_param *param);
1082:
1083: PTW32_DLLPORT int PTW32_CDECL pthread_getschedparam (pthread_t thread,
1084:          int *policy,
1085:          struct sched_param *param);
1086:
1087: PTW32_DLLPORT int PTW32_CDECL pthread_setconcurrency (int);
1088:
1089: PTW32_DLLPORT int PTW32_CDECL pthread_getconcurrency (void);
1090:
1091: /*
1092:  * Read-Write Lock Functions
1093:  */
1094: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_init(pthread_rwlock_t *lock,
1095:          const pthread_rwlockattr_t *attr);
1096:
1097: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_destroy(pthread_rwlock_t *lock);

```



```

1098:
1099: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_tryrdlock(pthread_rwlock_t *);
1100:
1101: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_trywrlock(pthread_rwlock_t *);
1102:
1103: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_rdlock(pthread_rwlock_t *lock);
1104:
1105: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_timedrdlock(pthread_rwlock_t *lock,
1106:                  const struct timespec *abstime);
1107:
1108: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_wrlock(pthread_rwlock_t *lock);
1109:
1110: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_timedwrlock(pthread_rwlock_t *lock,
1111:                  const struct timespec *abstime);
1112:
1113: PTW32_DLLPORT int PTW32_CDECL pthread_rwlock_unlock(pthread_rwlock_t *lock);
1114:
1115: PTW32_DLLPORT int PTW32_CDECL pthread_rwlockattr_init (pthread_rwlockattr_t * attr);
1116:
1117: PTW32_DLLPORT int PTW32_CDECL pthread_rwlockattr_destroy (pthread_rwlockattr_t * attr);
1118:
1119: PTW32_DLLPORT int PTW32_CDECL pthread_rwlockattr_getpshared (const
pthread_rwlockattr_t * attr,
1120:                  int *pshared);
1121:
1122: PTW32_DLLPORT int PTW32_CDECL pthread_rwlockattr_setpshared (pthread_rwlockattr_t *
attr,
1123:                  int pshared);
1124:
1125: #if PTW32_LEVEL >= PTW32_LEVEL_MAX - 1
1126:
1127: /*
1128:  * Signal Functions. Should be defined in <signal.h> but MSVC and MinGW32
1129:  * already have signal.h that don't define these.
1130:  */
1131: PTW32_DLLPORT int PTW32_CDECL pthread_kill(pthread_t thread, int sig);
1132:
1133: /*
1134:  * Non-portable functions
1135:  */
1136:

```

```

1137: /*
1138:  * Compatibility with Linux.
1139:  */
1140: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_setkind_np(pthread_mutexattr_t *
attr,
1141:                      int kind);
1142: PTW32_DLLPORT int PTW32_CDECL pthread_mutexattr_getkind_np(pthread_mutexattr_t *
attr,
1143:                      int *kind);
1144:
1145: /*
1146:  * Possibly supported by other POSIX threads implementations
1147:  */
1148: PTW32_DLLPORT int PTW32_CDECL pthread_delay_np (struct timespec * interval);
1149: PTW32_DLLPORT int PTW32_CDECL pthread_num_processors_np(void);
1150:
1151: /*
1152:  * Useful if an application wants to statically link
1153:  * the lib rather than load the DLL at run-time.
1154:  */
1155: PTW32_DLLPORT int PTW32_CDECL pthread_win32_process_attach_np(void);
1156: PTW32_DLLPORT int PTW32_CDECL pthread_win32_process_detach_np(void);
1157: PTW32_DLLPORT int PTW32_CDECL pthread_win32_thread_attach_np(void);
1158: PTW32_DLLPORT int PTW32_CDECL pthread_win32_thread_detach_np(void);
1159:
1160: /*
1161:  * Features that are auto-detected at load/run time.
1162:  */
1163: PTW32_DLLPORT int PTW32_CDECL pthread_win32_test_features_np(int);
1164: enum ptw32_features {
1165:  PTW32_SYSTEM_INTERLOCKED_COMPARE_EXCHANGE = 0x0001, /* System provides
it. */
1166:  PTW32_ALERTABLE_ASYNC_CANCEL              = 0x0002 /* Can cancel blocked threads. */
1167: };
1168:
1169: /*
1170:  * Register a system time change with the library.
1171:  * Causes the library to perform various functions
1172:  * in response to the change. Should be called whenever
1173:  * the application's top level window receives a
1174:  * WM_TIMECHANGE message. It can be passed directly to
1175:  * pthread_create() as a new thread if desired.

```

```

1176: */
1177: PTW32_DLLPORT void * PTW32_CDECL pthread_timechange_handler_np(void *);
1178:
1179: #endif /*PTW32_LEVEL >= PTW32_LEVEL_MAX - 1 */
1180:
1181: #if PTW32_LEVEL >= PTW32_LEVEL_MAX
1182:
1183: /*
1184:  * Returns the Win32 HANDLE for the POSIX thread.
1185:  */
1186: PTW32_DLLPORT HANDLE PTW32_CDECL pthread_getw32threadhandle_np(pthread_t
thread);
1187:
1188:
1189: /*
1190:  * Protected Methods
1191:  *
1192:  * This function blocks until the given WIN32 handle
1193:  * is signaled or pthread_cancel had been called.
1194:  * This function allows the caller to hook into the
1195:  * PThreads cancel mechanism. It is implemented using
1196:  *
1197:  *      WaitForMultipleObjects
1198:  *
1199:  * on 'waitHandle' and a manually reset WIN32 Event
1200:  * used to implement pthread_cancel. The 'timeout'
1201:  * argument to TimedWait is simply passed to
1202:  * WaitForMultipleObjects.
1203:  */
1204: PTW32_DLLPORT int PTW32_CDECL pthreadCancelableWait (HANDLE waitHandle);
1205: PTW32_DLLPORT int PTW32_CDECL pthreadCancelableTimedWait (HANDLE waitHandle,
1206:      DWORD timeout);
1207:
1208: #endif /* PTW32_LEVEL >= PTW32_LEVEL_MAX */
1209:
1210: /*
1211:  * Thread-Safe C Runtime Library Mappings.
1212:  */
1213: #ifndef _UWIN
1214: # if defined(NEED_ERRNO)
1215: PTW32_DLLPORT int * PTW32_CDECL _errno( void );

```

```

1216: # else
1217: #  ifndef errno
1218: #    if (defined(_MT) || defined(_DLL))
1219: #      __declspec(dllimport) extern int * __cdecl _errno(void);
1220: #      define errno  (*_errno())
1221: #    endif
1222: #  endif
1223: # endif
1224: #endif
1225:
1226: /*
1227:  * WIN32 C runtime library had been made thread-safe
1228:  * without affecting the user interface. Provide
1229:  * mappings from the UNIX thread-safe versions to
1230:  * the standard C runtime library calls.
1231:  * Only provide function mappings for functions that
1232:  * actually exist on WIN32.
1233:  */
1234:
1235: #if !defined(__MINGW32__)
1236: #define strtok_r( _s, _sep, _lasts ) \
1237:     ( (*_lasts) = strtok( _s, (_sep) ) )
1238: #endif /* !__MINGW32__ */
1239:
1240: #define asctime_r( _tm, _buf ) \
1241:     ( strcpy( (_buf), asctime( (_tm) ) ), \
1242:       (_buf) )
1243:
1244: #define ctime_r( _clock, _buf ) \
1245:     ( strcpy( (_buf), ctime( (_clock) ) ), \
1246:       (_buf) )
1247:
1248: #define gmtime_r( _clock, _result ) \
1249:     ( (*_result) = *gmtime( (_clock) ), \
1250:       (_result) )
1251:
1252: #define localtime_r( _clock, _result ) \
1253:     ( (*_result) = *localtime( (_clock) ), \
1254:       (_result) )
1255:
1256: #define rand_r( _seed ) \

```

```

1257:      ( _seed == _seed? rand() : rand() )
1258:
1259:
1260: /*
1261:  * Some compiler environments don't define some things.
1262:  */
1263: #if defined(__BORLANDC__)
1264: # define _ftime ftime
1265: # define _timeb timeb
1266: #endif
1267:
1268: #ifdef __cplusplus
1269:
1270: /*
1271:  * Internal exceptions
1272:  */
1273: class ptw32_exception {};
1274: class ptw32_exception_cancel : public ptw32_exception {};
1275: class ptw32_exception_exit : public ptw32_exception {};
1276:
1277: #endif
1278:
1279: #if PTW32_LEVEL >= PTW32_LEVEL_MAX
1280:
1281: /* FIXME: This is only required if the library was built using SEH */
1282: /*
1283:  * Get internal SEH tag
1284:  */
1285: PTW32_DLLPORT DWORD PTW32_CDECL ptw32_get_exception_services_code(void);
1286:
1287: #endif /* PTW32_LEVEL >= PTW32_LEVEL_MAX */
1288:
1289: #ifndef PTW32_BUILD
1290:
1291: #ifdef __CLEANUP_SEH
1292:
1293: /*
1294:  * Redefine the SEH __except keyword to ensure that applications
1295:  * propagate our internal exceptions up to the library's internal handlers.
1296:  */
1297: #define __except( E ) \

```

```

1298:     __except( ( GetExceptionCode() == ptw32_get_exception_services_code() ) \
1299:               ? EXCEPTION_CONTINUE_SEARCH : ( E ) )
1300:
1301: #endif /* __CLEANUP_SEH */
1302:
1303: #ifdef __CLEANUP_CXX
1304:
1305: /*
1306:  * Redefine the C++ catch keyword to ensure that applications
1307:  * propagate our internal exceptions up to the library's internal handlers.
1308:  */
1309: #ifdef _MSC_VER
1310:     /*
1311:      * WARNING: Replace any 'catch( ... )' with 'PtW32CatchAll'
1312:      * if you want Pthread-Win32 cancelation and pthread_exit to work.
1313:      */
1314:
1315: #ifndef PtW32NoCatchWarn
1316:
1317: #pragma message("Specify \"/DPTW32NoCatchWarn\" compiler flag to skip this message.")
1318: #pragma message("-----")
1319: #pragma message("When compiling applications with MSVC++ and C++ exception handling:")
1320: #pragma message(" Replace any 'catch( ... )' in routines called from POSIX threads")
1321: #pragma message(" with 'PtW32CatchAll' or 'CATCHALL' if you want POSIX thread")
1322: #pragma message(" cancelation and pthread_exit to work. For example:")
1323: #pragma message("")
1324: #pragma message(" #ifdef PtW32CatchAll")
1325: #pragma message("     PtW32CatchAll")
1326: #pragma message(" #else")
1327: #pragma message("     catch(...)")
1328: #pragma message(" #endif")
1329: #pragma message("     {")
1330: #pragma message("         /* Catchall block processing */")
1331: #pragma message("     }")
1332: #pragma message("-----")
1333:
1334: #endif
1335:
1336: #define PtW32CatchAll \
1337:     catch( ptw32_exception & ) { throw; } \
1338:     catch( ... )

```

```

1339:
1340: #else /* _MSC_VER */
1341:
1342: #define catch( E ) \
1343:     catch( ptw32_exception & ) { throw; } \
1344:     catch( E )
1345:
1346: #endif /* _MSC_VER */
1347:
1348: #endif /* __CLEANUP_CXX */
1349:
1350: #endif /* ! PTW32_BUILD */
1351:
1352: #ifdef __cplusplus
1353: } /* End of extern "C" */
1354: #endif /* __cplusplus */
1355:
1356: #ifdef PTW32__HANDLE_DEF
1357: # undef HANDLE
1358: #endif
1359: #ifdef PTW32__DWORD_DEF
1360: # undef DWORD
1361: #endif
1362:
1363: #undef PTW32_LEVEL
1364: #undef PTW32_LEVEL_MAX
1365:
1366: #endif /* ! RC_INVOKED */
1367:
1368: #ifdef interface
1369: # undef interface
1370: #endif
1371:
1372: #endif /* PTHREAD_H */

```

File: sdm/Win32/unix/stdint.h

```
1: #ifndef __stdint_h
2: #define __stdint_h
3:
4: typedef unsigned long uint32_t;
5: typedef long int32_t;
6: typedef unsigned short uint16_t;
7: typedef short int16_t;
8: typedef unsigned char uint8_t;
9: typedef char int8_t;
10:
11: #endif
```


File: sdm/Win32/unix/getopt.cpp

```
1: #include "getopt.h"
2:
3: /* Getopt for GNU.
4:  NOTE: getopt is now part of the C library, so if you don't know what
5:  "Keep this file name-space clean" means, talk to roland@gnu.ai.mit.edu
6:  before changing it!
7:
8:  Copyright (C) 1987, 88, 89, 90, 91, 92, 1993
9:    Free Software Foundation, Inc.
10:
11:  This program is free software; you can redistribute it and/or modify it
12:  under the terms of the GNU General Public License as published by the
13:  Free Software Foundation; either version 2, or (at your option) any
14:  later version.
15:
16:  This program is distributed in the hope that it will be useful,
17:  but WITHOUT ANY WARRANTY; without even the implied warranty of
18:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
19:  GNU General Public License for more details.
20:
21:  You should have received a copy of the GNU General Public License
22:  along with this program; if not, write to the Free Software
23:  Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.  */
24:
25: #ifdef __GNUC__
26:
27: #define alloca __builtin_alloca
28:
29: #else /* not __GNUC__ */
30: #if defined (HAVE_ALLOCA_H) || (defined(sparc) && (defined(sun) || (!defined(USG) &&
!defined(SVR4) && !defined(__svr4__))) || defined(WIN32)
31: #include <malloc.h>
32:
33: #else
34:
35: #ifndef _AIX
36: char *alloca ();
37: #endif
38:
```

```

39: #endif /* alloca.h */
40:
41: #endif /* not __GNU__ */
42:
43: #if !__STDC__ && !defined(const) && IN_GCC
44: #define const
45: #endif
46:
47: /* This tells Alpha OSF/1 not to define a getopt prototype in <stdio.h>. */
48: #ifndef _NO_PROTO
49: #define _NO_PROTO
50: #endif
51:
52: #include <stdio.h>
53:
54: /* Comment out all this code if we are using the GNU C Library, and are not
55:  actually compiling the library itself. This code is part of the GNU C
56:  Library, but also included in many other GNU distributions. Compiling
57:  and linking in this code is a waste when using the GNU C library
58:  (especially if it is a shared library). Rather than having every GNU
59:  program understand `config --with-gnu-libc' and omit the object files,
60:  it is simpler to just do this in the source for each such file. */
61:
62: #if defined (_LIBC) || !defined (__GNU_LIBRARY__)
63:
64:
65: /* This needs to come after some library #include
66:  to get __GNU_LIBRARY__ defined. */
67: #ifdef __GNU_LIBRARY__
68: #undef alloca
69: /* Don't include stdlib.h for non-GNU C libraries because some of them
70:  contain conflicting prototypes for getopt. */
71: #include <stdlib.h>
72: #else /* Not GNU C library. */
73: #define __alloca    alloca
74: #endif /* GNU C library. */
75:
76: /* If GETOPT_COMPAT is defined, `+' as well as `--' can introduce a
77:  long-named option. Because this is not POSIX.2 compliant, it is
78:  being phased out. */
79: /* #define GETOPT_COMPAT */

```

```

80:
81: /* This version of `getopt' appears to the caller like standard Unix `getopt'
82:  but it behaves differently for the user, since it allows the user
83:  to intersperse the options with the other arguments.
84:
85:  As `getopt' works, it permutes the elements of ARGV so that,
86:  when it is done, all the options precede everything else.  Thus
87:  all application programs are extended to handle flexible argument order.
88:
89:  Setting the environment variable POSIXLY_CORRECT disables permutation.
90:  Then the behavior is completely standard.
91:
92:  GNU application programs can use a third alternative mode in which
93:  they can distinguish the relative order of options and other arguments. */
94:
95: #include "getopt.h"
96:
97: /* For communication from `getopt' to the caller.
98:  When `getopt' finds an option that takes an argument,
99:  the argument value is returned here.
100:  Also, when `ordering' is RETURN_IN_ORDER,
101:  each non-option ARGV-element is returned here. */
102:
103: SDMLIB_API char *optarg = 0;
104:
105: /* Index in ARGV of the next element to be scanned.
106:  This is used for communication to and from the caller
107:  and for communication between successive calls to `getopt'.
108:
109:  On entry to `getopt', zero means this is the first call; initialize.
110:
111:  When `getopt' returns EOF, this is the index of the first of the
112:  non-option elements that the caller should itself scan.
113:
114:  Otherwise, `optind' communicates from one call to the next
115:  how much of ARGV has been scanned so far. */
116:
117: /* XXX 1003.2 says this must be 1 before any call. */
118: SDMLIB_API int optind = 0;
119:
120: /* The next char to be scanned in the option-element

```

```

121: in which the last option character we returned was found.
122: This allows us to pick up the scan where we left off.
123:
124: If this is zero, or a null string, it means resume the scan
125: by advancing to the next ARGV-element. */
126:
127: static char *nextchar;
128:
129: /* Callers store zero here to inhibit the error message
130: for unrecognized options. */
131:
132: int opterr = 1;
133:
134: /* Set to an option character which was unrecognized.
135: This must be initialized on some systems to avoid linking in the
136: system's own getopt implementation. */
137:
138: SDMLIB_API int optopt = '?';
139:
140: /* Describe how to deal with options that follow non-option ARGV-elements.
141:
142: If the caller did not specify anything,
143: the default is REQUIRE_ORDER if the environment variable
144: POSIXLY_CORRECT is defined, PERMUTE otherwise.
145:
146: REQUIRE_ORDER means don't recognize them as options;
147: stop option processing when the first non-option is seen.
148: This is what Unix does.
149: This mode of operation is selected by either setting the environment
150: variable POSIXLY_CORRECT, or using `+' as the first character
151: of the list of option characters.
152:
153: PERMUTE is the default. We permute the contents of ARGV as we scan,
154: so that eventually all the non-options are at the end. This allows options
155: to be given in any order, even with programs that were not written to
156: expect this.
157:
158: RETURN_IN_ORDER is an option available to programs that were written
159: to expect options and other ARGV-elements in any order and that care about
160: the ordering of the two. We describe each non-option ARGV-element
161: as if it were the argument of an option with character code 1.

```

```

162: Using '-' as the first character of the list of option characters
163: selects this mode of operation.
164:
165: The special argument '--' forces an end of option-scanning regardless
166: of the value of 'ordering'. In the case of RETURN_IN_ORDER, only
167: '--' can cause 'getopt' to return EOF with 'optind' != ARGV. */
168:
169: static enum
170: {
171:   REQUIRE_ORDER, PERMUTE, RETURN_IN_ORDER
172: } ordering;
173:
174: #if defined(__GNU_LIBRARY__) || (WIN32)
175: /* We want to avoid inclusion of string.h with non-GNU libraries
176:  because there are many ways it can cause trouble.
177:  On some systems, it contains special magic macros that don't work
178:  in GCC. */
179: #include <string.h>
180: #define my_index strchr
181: #define my_bcopy(src, dst, n) memcpy((dst), (src), (n))
182: #else
183:
184: /* Avoid depending on library functions or files
185:  whose names are inconsistent. */
186:
187: char *getenv ();
188:
189: static char *
190: my_index (const char* str, int chr)
191: {
192:   while (*str)
193:   {
194:     if (*str == chr)
195:       return (char *) str;
196:     str++;
197:   }
198:   return 0;
199: }
200:
201: static void
202: my_bcopy (const char* from, char* to, int size)

```

```

203: {
204:   int i;
205:   for (i = 0; i < size; i++)
206:     to[i] = from[i];
207: }
208: #endif          /* GNU C library. */
209:
210: /* Handle permutation of arguments. */
211:
212: /* Describe the part of ARGV that contains non-options that have
213:    been skipped. `first_nonopt' is the index in ARGV of the first of them;
214:    `last_nonopt' is the index after the last of them. */
215:
216: static int first_nonopt;
217: static int last_nonopt;
218:
219: /* Exchange two adjacent subsequences of ARGV.
220:    One subsequence is elements [first_nonopt,last_nonopt)
221:    which contains all the non-options that have been skipped so far.
222:    The other is elements [last_nonopt,optind), which contains all
223:    the options processed since those non-options were skipped.
224:
225:    `first_nonopt' and `last_nonopt' are relocated so that they describe
226:    the new indices of the non-options in ARGV after they are moved. */
227:
228: static void
229: exchange (char** argv)
230: {
231:   int nonopts_size = (last_nonopt - first_nonopt) * sizeof (char *);
232:   char **temp = (char **) __alloca (nonopts_size);
233:
234:   /* Interchange the two blocks of data in ARGV. */
235:
236:   my_bcopy ((char *) &argv[first_nonopt], (char *) temp, nonopts_size);
237:   my_bcopy ((char *) &argv[last_nonopt], (char *) &argv[first_nonopt],
238:             (optind - last_nonopt) * sizeof (char *));
239:   my_bcopy ((char *) temp,
240:             (char *) &argv[first_nonopt + optind - last_nonopt],
241:             nonopts_size);
242:
243:   /* Update records for the slots the non-options now occupy. */

```

```

244:
245: first_nonopt += (optind - last_nonopt);
246: last_nonopt = optind;
247: }
248:
249: /* Scan elements of ARGV (whose length is ARGC) for option characters
250:    given in OPTSTRING.
251:
252:    If an element of ARGV starts with '-', and is not exactly "-" or "--",
253:    then it is an option element.  The characters of this element
254:    (aside from the initial '-') are option characters.  If `getopt'
255:    is called repeatedly, it returns successively each of the option characters
256:    from each of the option elements.
257:
258:    If `getopt' finds another option character, it returns that character,
259:    updating `optind' and `nextchar' so that the next call to `getopt' can
260:    resume the scan with the following option character or ARGV-element.
261:
262:    If there are no more option characters, `getopt' returns `EOF'.
263:    Then `optind' is the index in ARGV of the first ARGV-element
264:    that is not an option.  (The ARGV-elements have been permuted
265:    so that those that are not options now come last.)
266:
267:    OPTSTRING is a string containing the legitimate option characters.
268:    If an option character is seen that is not listed in OPTSTRING,
269:    return '?' after printing an error message.  If you set `opterr' to
270:    zero, the error message is suppressed but we still return '?'.
271:
272:    If a char in OPTSTRING is followed by a colon, that means it wants an arg,
273:    so the following text in the same ARGV-element, or the text of the following
274:    ARGV-element, is returned in `optarg'.  Two colons mean an option that
275:    wants an optional arg; if there is text in the current ARGV-element,
276:    it is returned in `optarg', otherwise `optarg' is set to zero.
277:
278:    If OPTSTRING starts with `-' or `+', it requests different methods of
279:    handling the non-option ARGV-elements.
280:    See the comments about RETURN_IN_ORDER and REQUIRE_ORDER, above.
281:
282:    Long-named options begin with `--' instead of `-'.
```

283: Their names may be abbreviated as long as the abbreviation is unique
 284: or is an exact match for some defined option. If they have an

```

285: argument, it follows the option name in the same ARGV-element, separated
286: from the option name by a '=', or else the in next ARGV-element.
287: When `getopt' finds a long-named option, it returns 0 if that option's
288: `flag' field is nonzero, the value of the option's `val' field
289: if the `flag' field is zero.
290:
291: The elements of ARGV aren't really const, because we permute them.
292: But we pretend they're const in the prototype to be compatible
293: with other systems.
294:
295: LONGOPTS is a vector of `struct option' terminated by an
296: element containing a name which is zero.
297:
298: LONGIND returns the index in LONGOPT of the long-named option found.
299: It is only valid when a long-named option has been found by the most
300: recent call.
301:
302: If LONG_ONLY is nonzero, '-' as well as '--' can introduce
303: long-named options. */
304:
305: int
306: _getopt_internal (int argc, char *const *argv, const char* optstring, const struct option* longopts,
307:   int* longind, int long_only)
308: {
309:   int option_index;
310:
311:   optarg = 0;
312:
313:   /* Initialize the internal data when the first call is made.
314:    Start processing options with ARGV-element 1 (since ARGV-element 0
315:    is the program name); the sequence of previously skipped
316:    non-option ARGV-elements is empty. */
317:
318:   if (optind == 0)
319:   {
320:     first_nonopt = last_nonopt = optind = 1;
321:
322:     nextchar = NULL;
323:
324:     /* Determine how to handle the ordering of options and nonoptions. */
325:

```



```

326:   if (optstring[0] == '-')
327:   {
328:       ordering = RETURN_IN_ORDER;
329:       ++optstring;
330:   }
331:   else if (optstring[0] == '+')
332:   {
333:       ordering = REQUIRE_ORDER;
334:       ++optstring;
335:   }
336: //   else if (getenv ("POSIXLY_CORRECT") != NULL)
337: // ordering = REQUIRE_ORDER;
338:   else
339:   ordering = PERMUTE;
340:   }
341:
342: if (nextchar == NULL || *nextchar == '\0')
343:   {
344:       if (ordering == PERMUTE)
345:       {
346:           /* If we have just processed some options following some non-options,
347:              exchange them so that the options come first. */
348:
349:           if (first_nonopt != last_nonopt && last_nonopt != optind)
350:               exchange ((char **) argv);
351:           else if (last_nonopt != optind)
352:               first_nonopt = optind;
353:
354:           /* Now skip any additional non-options
355:              and extend the range of non-options previously skipped. */
356:
357:           while (optind < argc
358:                  && (argv[optind][0] != '-' || argv[optind][1] == '\0'))
359: #ifdef GETOPT_COMPAT
360:                 && (longopts == NULL
361:                     || argv[optind][0] != '+' || argv[optind][1] == '\0'))
362: #endif
363:                 /* GETOPT_COMPAT */
364:                 optind++;
365:           last_nonopt = optind;
366:       }

```

```

367:
368:  /* Special ARGV-element '--' means premature end of options.
369:  Skip it like a null option,
370:  then exchange with previous non-options as if it were an option,
371:  then skip everything else like a non-option. */
372:
373:  if (optind != argc && !strcmp (argv[optind], "--"))
374:  {
375:    optind++;
376:
377:    if (first_nonopt != last_nonopt && last_nonopt != optind)
378:      exchange ((char **) argv);
379:    else if (first_nonopt == last_nonopt)
380:      first_nonopt = optind;
381:    last_nonopt = argc;
382:
383:    optind = argc;
384:  }
385:
386:  /* If we have done all the ARGV-elements, stop the scan
387:  and back over any non-options that we skipped and permuted. */
388:
389:  if (optind == argc)
390:  {
391:    /* Set the next-arg-index to point at the non-options
392:    that we previously skipped, so the caller will digest them. */
393:    if (first_nonopt != last_nonopt)
394:      optind = first_nonopt;
395:    return EOF;
396:  }
397:
398:  /* If we have come to a non-option and did not permute it,
399:  either stop the scan or describe it to the caller and pass it by. */
400:
401:  if ((argv[optind][0] != '-' || argv[optind][1] == '\0')
402:  #ifdef GETOPT_COMPAT
403:    && (longopts == NULL
404:    || argv[optind][0] != '+' || argv[optind][1] == '\0')
405:  #endif
    /* GETOPT_COMPAT */
406:  )
407:  {

```

```

408:     if (ordering == REQUIRE_ORDER)
409:         return EOF;
410:     optarg = argv[optind++];
411:     return 1;
412: }
413:
414: /* We have found another option-ARGV-element.
415:  Start decoding its characters. */
416:
417:     nextchar = (argv[optind] + 1
418:         + (longopts != NULL && argv[optind][1] == '-'));
419: }
420:
421: if (longopts != NULL
422:     && ((argv[optind][0] == '-'
423:         && (argv[optind][1] == '-' || long_only))
424: #ifdef GETOPT_COMPAT
425:     || argv[optind][0] == '+')
426: #endif
427:     ))
428: {
429:     const struct option *p;
430:     char *s = nextchar;
431:     int exact = 0;
432:     int ambig = 0;
433:     const struct option *pfound = NULL;
434:     int indfound;
435:
436:     while (*s && *s != '=')
437:         s++;
438:
439:     /* Test all options for either exact match or abbreviated matches. */
440:     for (p = longopts, option_index = 0; p->name;
441:         p++, option_index++)
442:         if (!strcmp(p->name, nextchar, s - nextchar))
443:             {
444:                 if (s - nextchar == (int)strlen(p->name))
445:                     {
446:                         /* Exact match found. */
447:                         pfound = p;
448:                         indfound = option_index;

```

```

449:     exact = 1;
450:     break;
451: }
452: else if (pfound == NULL)
453: {
454:     /* First nonexact match found. */
455:     pfound = p;
456:     indfound = option_index;
457: }
458: else
459:     /* Second nonexact match found. */
460:     ambig = 1;
461: }
462:
463: if (ambig && !exact)
464: {
465:     if (opterr)
466:         fprintf (stderr, "%s: option '%s' is ambiguous \n",
467:                 argv[0], argv[optind]);
468:     nextchar += strlen (nextchar);
469:     optind++;
470:     return '?';
471: }
472:
473: if (pfound != NULL)
474: {
475:     option_index = indfound;
476:     optind++;
477:     if (*s)
478:     {
479:         /* Don't test has_arg with >, because some C compilers don't
480:            allow it to be used on enums. */
481:         if (pfound->has_arg)
482:             optarg = s + 1;
483:         else
484:         {
485:             if (opterr)
486:             {
487:                 if (argv[optind - 1][1] == '-')
488:                     /* --option */
489:                     fprintf (stderr,

```

```

490:         "%s: option `--%s' doesn't allow an argument \n",
491:         argv[0], pfound->name);
492:     else
493:         /* +option or -option */
494:         fprintf (stderr,
495:             "%s: option `%c%s' doesn't allow an argument \n",
496:             argv[0], argv[optind - 1][0], pfound->name);
497:     }
498:     nextchar += strlen (nextchar);
499:     return '?';
500: }
501: }
502: else if (pfound->has_arg == 1)
503: {
504:     if (optind < argc)
505:         optarg = argv[optind++];
506:     else
507:     {
508:         if (opterr)
509:             fprintf (stderr, "%s: option `%s' requires an argument \n",
510:                 argv[0], argv[optind - 1]);
511:         nextchar += strlen (nextchar);
512:         return optstring[0] == ':' ? ':' : '?';
513:     }
514: }
515: nextchar += strlen (nextchar);
516: if (longind != NULL)
517:     *longind = option_index;
518: if (pfound->flag)
519: {
520:     *(pfound->flag) = pfound->val;
521:     return 0;
522: }
523: return pfound->val;
524: }
525: /* Can't find it as a long option.  If this is not getopt_long_only,
526: or the option starts with '--' or is not a valid short
527: option, then it's an error.
528: Otherwise interpret it as a short option.  */
529: if (!long_only || argv[optind][1] == '-')
530: #ifdef GETOPT_COMPAT

```

```

531:     || argv[optind][0] == '+'
532: #endif          /* GETOPT_COMPAT */
533:     || my_index (optstring, *nextchar) == NULL)
534: {
535:     if (opterr)
536:     {
537:         if (argv[optind][1] == '-')
538:             /* --option */
539:             fprintf (stderr, "%s: unrecognized option `--%s'\n",
540:                     argv[0], nextchar);
541:         else
542:             /* +option or -option */
543:             fprintf (stderr, "%s: unrecognized option `-%c%s'\n",
544:                     argv[0], argv[optind][0], nextchar);
545:     }
546:     nextchar = (char *) "";
547:     optind++;
548:     return '?';
549: }
550: }
551:
552: /* Look at and handle the next option-character. */
553:
554: {
555:     char c = *nextchar++;
556:     char *temp = (char*)my_index (optstring, c);
557:
558:     /* Increment `optind' when we start to process its last character. */
559:     if (*nextchar == '\0')
560:         ++optind;
561:
562:     if (temp == NULL || c == ':')
563:     {
564:         if (opterr)
565:         {
566: #if 0
567:             if (c < 040 || c >= 0177)
568:                 fprintf (stderr, "%s: unrecognized option, character code 0%o\n",
569:                         argv[0], c);
570:             else
571:                 fprintf (stderr, "%s: unrecognized option `-%c'\n", argv[0], c);

```

```

572: #else
573:     /* 1003.2 specifies the format of this message. */
574:     fprintf (stderr, "%s: illegal option -- %c \n", argv[0], c);
575: #endif
576: }
577: optopt = c;
578: return '?';
579: }
580: if (temp[1] == ':')
581: {
582:     if (temp[2] == ':')
583:     {
584:         /* This is an option that accepts an argument optionally. */
585:         if (*nextchar != '\0')
586:         {
587:             optarg = nextchar;
588:             optind++;
589:         }
590:         else
591:             optarg = 0;
592:         nextchar = NULL;
593:     }
594:     else
595:     {
596:         /* This is an option that requires an argument. */
597:         if (*nextchar != '\0')
598:         {
599:             optarg = nextchar;
600:             /* If we end this ARGV-element by taking the rest as an arg,
601:              we must advance to the next element now. */
602:             optind++;
603:         }
604:         else if (optind == argc)
605:         {
606:             if (opterr)
607:             {
608: #if 0
609:                 fprintf (stderr, "%s: option `-%c' requires an argument \n",
610:                     argv[0], c);
611: #else
612:                 /* 1003.2 specifies the format of this message. */

```

```

613:         fprintf(stderr, "%s: option requires an argument -- %c \n",
614:                 argv[0], c);
615: #endif
616:     }
617:     optopt = c;
618:     if (optstring[0] == ':')
619:         c = ':';
620:     else
621:         c = '?';
622:     }
623: else
624:     /* We already incremented `optind' once;
625:      increment it again when taking next ARGV-elt as argument. */
626:     optarg = argv[optind++];
627:     nextchar = NULL;
628: }
629: }
630: return c;
631: }
632: }
633:
634: SDMLIB_API int getopt (int argc, char *const *argv, const char* optstring)
635: {
636:     return _getopt_internal (argc, argv, optstring,
637:                             (const struct option *) 0,
638:                             (int *) 0,
639:                             0);
640: }
641:
642: SDMLIB_API int getopt_long (int argc, char *const *argv, const char* optstring,
643:                             const struct option *longopts, int *longind)
644: {
645:     return _getopt_internal (argc, argv, optstring,
646:                             longopts, longind, 0);
647: }
648:
649: SDMLIB_API int getopt_long_only (int argc, char *const *argv, const char* optstring,
650:                                 const struct option *longopts, int *longind)
651: {
652:     return _getopt_internal (argc, argv, optstring,
653:                             longopts, longind, 1);

```



```

654: }
655:
656: #endif /* _LIBC or not __GNU_LIBRARY__. */
657:
658: #ifdef TEST
659:
660: /* Compile with -DTEST to make an executable for use in testing
661:  the above definition of `getopt'. */
662:
663: int
664: main (int argc, char** argv)
665: {
666:  int c;
667:  int digit_optind = 0;
668:
669:  while (1)
670:  {
671:    int this_option_optind = optind ? optind : 1;
672:
673:    c = getopt (argc, argv, "abc:d:0123456789");
674:    if (c == EOF)
675:      break;
676:
677:    switch (c)
678:    {
679:      case '0':
680:      case '1':
681:      case '2':
682:      case '3':
683:      case '4':
684:      case '5':
685:      case '6':
686:      case '7':
687:      case '8':
688:      case '9':
689:        if (digit_optind != 0 && digit_optind != this_option_optind)
690:          printf ("digits occur in two different argv-elements. \n");
691:        digit_optind = this_option_optind;
692:        printf ("option %c \n", c);
693:        break;
694:

```

```

695:  case 'a':
696:      printf ("option a \n");
697:      break;
698:
699:  case 'b':
700:      printf ("option b \n");
701:      break;
702:
703:  case 'c':
704:      printf ("option c with value `%'s' \n", optarg);
705:      break;
706:
707:  case '?':
708:      break;
709:
710:  default:
711:      printf ("?? getopt returned character code 0%o ?? \n", c);
712:  }
713: }
714:
715: if (optind < argc)
716: {
717:     printf ("non-option ARGV-elements: ");
718:     while (optind < argc)
719:         printf ("%s ", argv[optind++]);
720:     printf ("\n");
721: }
722:
723: exit (0);
724: }
725:
726: #endif /* TEST */

```

File: sdm/Win32/unix/unistd.h

```
1: #ifndef __unistd_h_
2: #define __unistd_h_
3:
4: #include "stdlib.h"
5: #include <unix/fcntl.h>
6: #include "sdmLib.h"
7:
8: // #define main AppMain
9:
10: typedef unsigned short ushort;
11: typedef unsigned int uint;
12: typedef unsigned long ulong;
13:
14: typedef unsigned short u_int16_t;
15: typedef unsigned int u_int32_t;
16:
17: typedef int key_t;
18: typedef int pid_t;
19: typedef int uid_t;
20:
21: // Missing defines for open
22: #ifndef S_IRUSR
23: #define S_IRUSR 0400
24: #define S_IWUSR 0200
25: #define S_IXUSR 0100
26: #endif
27: #ifndef S_IRGRP
28: #define S_IRGRP 0040
29: #define S_IWGRP 0020
30: #define S_IXGRP 0010
31: #endif
32: #ifndef S_IROTH
33: #define S_IROTH 0004
34: #define S_IWOTH 0002
35: #define S_IXOTH 0001
36: #endif
37:
38: #define chmod wChmod
39: #define close wClose
```

```

40: #define read wRead
41: #define write wWrite
42: #define lseek wLseek
43: #define snprintf _snprintf
44:
45: extern SDMLIB_API int fcntl(int fd, uint flags);
46: extern SDMLIB_API int fcntl(int fd, uint op, uint flags);
47: extern SDMLIB_API int open(const char* filepath, unsigned int flags);
48:
49: #ifdef __cplusplus
50: extern "C" {
51: #endif
52:
53: extern SDMLIB_API void chdir(const char* newDirectory);
54: extern SDMLIB_API int chmod(const char* filename, int mode);
55: extern SDMLIB_API int close(int fd);
56: extern SDMLIB_API pid_t fork(void);
57: extern SDMLIB_API int ioctl(int fd, uint op, ...);
58: extern SDMLIB_API int pipe(int fd[2]);
59: extern SDMLIB_API int read(int fd, void* buffer, int len);
60: extern SDMLIB_API void sleep(uint seconds);
61: extern SDMLIB_API char* strerror_r(int errornum, char* buf, size_t len);
62: extern SDMLIB_API char* strdup(const char* str, size_t len);
63: extern SDMLIB_API void usleep( uint microseconds );
64: extern SDMLIB_API int write(int fd, const void* buffer, int len);
65: extern SDMLIB_API int lseek(int fd, long offset, int origin);
66:
67: #ifdef __cplusplus
68: }
69: #endif
70:
71: #endif // __unistd_h_

```

File: sdm/Win32/unix/unix.cpp

```
1: // This is the main DLL file.
2: #include <unistd.h>
3: #define WIN32_LEAN_AND_MEAN
4: #include <windows.h>
5: #include <stdio.h>
6: #include <stdarg.h>
7: #include <errno.h>
8: #include <pthread.h>
9:
10: #undef chmod
11: #undef close
12: #undef read
13: #undef write
14: #undef lseek
15:
16: #include <io.h>
17: #include <unix/fcntl.h>
18:
19: #include <fcntl.h>
20: #include <sys/types.h>
21: #include <sys/stat.h>
22:
23: struct FilePair
24: {
25:     char* name;
26:     int reader, writer;
27:     int timeout, length;
28:     bool versionSent;
29: };
30: static int numFilePairs = 1;
31: static struct FilePair** filePairs;
32: #define FILE_PAIR_BASE 10000
33: pthread_mutex_t fileTableMutex = PTHREAD_MUTEX_INITIALIZER;
34:
35: SDMLIB_API void chdir( const char* newDirectory )
36: {
37:     SetCurrentDirectory(newDirectory);
38: }
39:
```

```

40: SDMLIB_API int wChmod( const char* filename, int mode )
41: {
42:     return _chmod(filename,mode);
43: }
44:
45:
46: SDMLIB_API int wClose( int fd )
47: {
48:     if ( fd > FILE_PAIR_BASE )
49:     {
50:         pthread_mutex_lock( &fileTableMutex );
51:         if ( fd-FILE_PAIR_BASE >= numFilePairs )
52:         {
53:             errno = EINVAL;
54:             pthread_mutex_unlock( &fileTableMutex );
55:             return -1;
56:         }
57:         struct FilePair* fp = filePairs[fd-FILE_PAIR_BASE];
58:         pthread_mutex_unlock ( &fileTableMutex );
59:         if( fp->reader > 0 )
60:         {
61:             _close( fp->reader);
62:         }
63:         int status = 0;
64:         if(fp->writer > 0)
65:         {
66:             status = _close(fp->writer);
67:         }
68:         free(fp->name);
69:         fp->name = NULL;
70:         fp->reader = fp->writer = -1;
71:         return status;
72:     }
73:     return _close(fd);
74: }
75:
76: SDMLIB_API int fcntl( int fd, unsigned int op, unsigned int flags)
77: {
78:     if ( op != F_SETFL || flags != O_NONBLOCK )
79:         printf("fcntl(%d,%d,0x%0x) \n",fd,op,flags);
80:     return 0;

```

```

81: }
82:
83: SDMLIB_API int fcntl( int fd, unsigned int flags)
84: {
85: if ( flags != O_NONBLOCK )
86:     printf("fcntl(%d,0x%0x) \n",fd,flags);
87: return 0;
88: }
89:
90: static int pid = 77;
91: SDMLIB_API pid_t fork()
92: {
93: printf("fork() = %d \n",pid);
94: return pid++;
95: }
96:
97: SDMLIB_API int ioctl(int fd, uint op, ...)
98: {
99: int value;
100:     DWORD returnedLength;
101:     char outBuf[16];
102:     va_list marker;
103:     va_start( marker, op );    /* Initialize variable arguments. */
104:     value = va_arg( marker, int);
105:     va_end( marker );         /* Reset variable arguments.    */
106: // printf("ioctl(%d,%d,0x%0x) \n", fd, op, value);
107: // *(int*)inBuf = value;
108: if ( fd > FILE_PAIR_BASE )
109: {
110:     if ( fd-FILE_PAIR_BASE >= numFilePairs )
111:     {
112:         errno = EINVAL;
113:         pthread_mutex_unlock( &fileTableMutex );
114:         return -1;
115:     }
116:     if ( op & 0xC000 )
117:     {
118:         struct FilePair* fp = filePairs[fd-FILE_PAIR_BASE];
119:         pthread_mutex_unlock( &fileTableMutex );
120:         switch ( op )
121:         {

```

```

122:         case 0xC001:
123:             *(int*)value = 1;
124:             break;
125:         case 0xC002:
126:             fp->timeout = value;
127:             break;
128:         case 0xC003:
129:             printf(" \nDon't understand RobustHub command for %s \n \n", fp->name);
130:             break;
131:         case 0xC004:
132:             char* path = (char*)value;
133:             int port = fp->name[strlen(fp->name)-1] - '0';
134:             sprintf(path,"port %d.%d", port/4,port%4);
135:             break;
136:     }
137: }
138: }
139: BOOL status = DeviceIoControl( (HANDLE)fd, op, (void*)&value, 4, outBuf, sizeof(outBuf),
&returnedLength, NULL);
140: // value = *(int*)outBuf;
141: return 0;
142: }
143:
144: SDMLIB_API int open( const char* filepath, unsigned int flags)
145: {
146:     if ( strcmp(filepath,"/dev/asim",9) == 0 )
147:     {
148:         struct FilePair* fp = NULL;
149:         pthread_mutex_lock( &fileTableMutex );
150:         int fd;
151:         for( fd=1; fd < numFilePairs; fd++ )
152:         {
153:             if ( filePairs[fd] != NULL && filePairs[fd]->name == NULL )
154:             {
155:                 fp = filePairs[fd];
156:                 fd += FILE_PAIR_BASE;
157:                 break;
158:             }
159:         }
160:         if ( fp == NULL )
161:         {

```



```

162:         fp = (struct FilePair*)calloc(sizeof(struct FilePair),1);
163:         filePairs = (struct FilePair**)realloc( filePairs, (numFilePairs+1)*sizeof(struct
FilePair*));
164:         filePairs[numFilePairs] = fp;
165:         fd = FILE_PAIR_BASE+numFilePairs++;
166:     }
167:     fp->name = strdup(filepath);
168:     pthread_mutex_unlock( &fileTableMutex );
169:     char name[64];
170:     strcpy(name,filepath);
171:     strcat(name, ".r");
172:     fp->reader = _open(name,flags|_O_BINARY);
173:     strcpy(name,filepath);
174:     strcat(name, ".w");
175:     fp->writer = _open(name,flags|_O_BINARY);
176:     return fd;
177: }
178: return _open(filepath,flags,_S_IREAD | _S_IWRITE);
179: }
180:
181: SDMLIB_API int pipe( int fd[2])
182: {
183:     return _pipe( fd, 256*1024, _O_BINARY );
184: }
185:
186: SDMLIB_API int wRead(int fd, void* buf, int len)
187: {
188:     if ( fd > FILE_PAIR_BASE )
189:     {
190:         pthread_mutex_lock( &fileTableMutex );
191:         if ( fd-FILE_PAIR_BASE >= numFilePairs )
192:         {
193:             errno = EINVAL;
194:             pthread_mutex_unlock( &fileTableMutex );
195:             return -1;
196:         }
197:         struct FilePair* fp = filePairs[fd-FILE_PAIR_BASE];
198:         pthread_mutex_unlock( &fileTableMutex );
199:         if ( fp->reader == -1 ) return 0;
200:         char buffer[70];
201:         int i, offset, n;

```

```

202:      char* chars = (char*)buf;
203: AGAIN:
204:      buffer[0] = buffer[1] = buffer[2] = 0;
205:      do {
206:          n = _read(fp->reader,buffer,6);
207:          chars[0] = buffer[0];
208:          if ( n > 3 )
209:          {
210:              if ( '0' <= buffer[1] && buffer[1] <= '9'
211:                  && '0' <= buffer[2] && buffer[2] <= '9'
212:                  && '0' <= buffer[3] && buffer[3] <= '9'
213:                  && '0' <= buffer[4] && buffer[4] <= '9'
214:                  && '0' <= buffer[5] && buffer[5] <= '9' )
215:              {
216:                  sscanf(&buffer[1], "%5d", &fp->length );
217:
218:                  if ( fp->versionSent )
219:                  {
220:                      chars[1] = fp->length/256;
221:                      chars[2] = fp->length%256;
222:                  }
223:                  else
224:                  {
225:                      chars[2] = fp->length/256;
226:                      chars[1] = fp->length%256;
227:                  }
228:                  offset = 3;
229:                  switch ( chars[0] )
230:                  {
231:                      case 'T':
232:                          Sleep(fp->length);
233:                          fp->length = 0;
234:                          break;
235:                      case 'V':
236:                          fp->versionSent = true;
237:                          break;
238:                      default:
239:                          break;
240:                  }
241:              }
242:          } else

```

```

243:         {
244:             offset = 6;
245:             for( i=0; i<offset; i++)
246:                 chars[i] = buffer[i];
247:         }
248:
249:         if ( fp->length >= 0 )
250:         {
251:             int m = fp->length+2;
252:             if ( len-offset < fp->length )
253:                 m = len-offset;
254:             n = _read(fp->reader,&chars[offset],m) + offset;
255:             if ( chars[0] == 'T' )
256:                 goto AGAIN;
257:         }
258:         else
259:             n = offset;
260:         return n;
261:     }
262:     if ( n == 0 )
263:         Sleep(1000);
264: } while ( n == 0 );
265: return n;
266: }
267: return _read(fd, buf, len);
268: }
269:
270: SDMLIB_API void sleep( unsigned int seconds )
271: {
272:     Sleep(seconds*1000);
273: }
274:
275: SDMLIB_API char* strerror_r(int errornum, char* buf, size_t len)
276: {
277:     int error = *_errno();
278:
279:     if ( error < _sys_nerr )
280:         strncpy(buf,_sys_errlist[error],len);
281:     else
282:         sprintf(buf,"errno, %d, out of range, %d",error,_sys_nerr);
283:     return buf;

```

```

284: }
285:
286: SDMLIB_API char* strndup( const char* str, size_t n )
287: {
288:     size_t len = strlen(str);
289:     if ( len > n )
290:     {
291:         char* dup = (char*)malloc(n+1);
292:         strncpy(dup,str,n);
293:         dup[n] = 0;
294:         return dup;
295:     }
296:     else
297:         return strdup(str);
298: }
299:
300: SDMLIB_API void usleep(unsigned int microsec)
301: {
302:     if ( microsec%1000 != 0 )
303:     {
304:         //printf("usleep(%d) \n",microsec);
305:     }
306:     Sleep(microsec/1000);
307: }
308:
309: SDMLIB_API int wait( int* status )
310: {
311:     *status = 0;
312:     printf("wait(%d) \n",*status);
313:     return 0;
314: }
315:
316: SDMLIB_API int wWrite(int fd, const void* buf, int len)
317: {
318:     if ( fd > FILE_PAIR_BASE )
319:     {
320:         pthread_mutex_lock( &fileTableMutex );
321:         if ( fd-FILE_PAIR_BASE >= numFilePairs )
322:         {
323:             errno = EINVAL;
324:             pthread_mutex_unlock( &fileTableMutex );

```

```

325:         return -1;
326:     }
327:     struct FilePair* fp = filePairs[fd-FILE_PAIR_BASE];
328:     pthread_mutex_unlock( &fileTableMutex );
329:     if ( fp->writer == -1 ) return 0;
330:     fd = fp->writer;
331: }
332: return _write(fd, buf, len);
333: }
334:
335: SDMLIB_API int wLseek(int _FileHandle, long _Offset, int _Origin)
336: {
337:     return _lseek(_FileHandle,_Offset,_Origin);
338: }

```

File: sdm/Win32/unix/getopt.h

```
1: #ifndef __getopt_h_
2: #define __getopt_h_
3:
4: #include "sdmLib.h"
5:
6: /* For communication from `getopt' to the caller.
7:  When `getopt' finds an option that takes an argument,
8:  the argument value is returned here.
9:  Also, when `ordering' is RETURN_IN_ORDER,
10:  each non-option ARGV-element is returned here. */
11:
12: extern SDMLIB_API char *optarg;
13:
14: /* Index in ARGV of the next element to be scanned.
15:  This is used for communication to and from the caller
16:  and for communication between successive calls to `getopt'.
17:
18:  On entry to `getopt', zero means this is the first call; initialize.
19:
20:  When `getopt' returns EOF, this is the index of the first of the
21:  non-option elements that the caller should itself scan.
22:
23:  Otherwise, `optind' communicates from one call to the next
24:  how much of ARGV has been scanned so far. */
25:
26: extern SDMLIB_API int optind;
27:
28: /* Callers store zero here to inhibit the error message `getopt' prints
29:  for unrecognized options. */
30:
31: extern SDMLIB_API int opterr;
32:
33: /* Set to an option character which was unrecognized. */
34:
35: extern SDMLIB_API int optopt;
36:
37: /* Describe the long-named options requested by the application.
38:  The LONG_OPTIONS argument to getopt_long or getopt_long_only is a vector
39:  of `struct option' terminated by an element containing a name which is
```

```

40: zero.
41:
42: The field `has_arg' is:
43: no_argument      (or 0) if the option does not take an argument,
44: required_argument (or 1) if the option requires an argument,
45: optional_argument (or 2) if the option takes an optional argument.
46:
47: If the field `flag' is not NULL, it points to a variable that is set
48: to the value given in the field `val' when the option is found, but
49: left unchanged if the option is not found.
50:
51: To have a long-named option do something other than set an `int' to
52: a compiled-in constant, such as set a value from `optarg', set the
53: option's `flag' field to zero and its `val' field to a nonzero
54: value (the equivalent single-letter option character, if there is
55: one). For long options that have a zero `flag' field, `getopt'
56: returns the contents of the `val' field. */
57:
58: struct option
59: {
60:     const char *name;
61:     /* has_arg can't be an enum because some compilers complain about
62:        type mismatches in all the code that assumes it is an int. */
63:     int has_arg;
64:     int *flag;
65:     int val;
66: };
67:
68: /* Names for the values of the `has_arg' field of `struct option'. */
69:
70: #define no_argument
71:
72: #define required_argument 1
73: #define optional_argument 2
74:
75: extern SDMLIB_API int getopt (int argc, char *const *argv, const char *shortopts);
76: extern SDMLIB_API int getopt_long (int argc, char *const *argv, const char *shortopts,
77:                                     const struct option *longopts, int *longind);
78: extern SDMLIB_API int getopt_long_only (int argc, char *const *argv,
79:                                         const char *shortopts,
80:                                         const struct option *longopts, int *longind);

```

```
81:
82: /* Internal only. Users should not call this directly. */
83: extern int __getopt_internal (int argc, char *const *argv,
84:         const char *shortopts,
85:         const struct option *longopts, int *longind,
86:         int long_only);
87:
88: #endif // __getopt_h_
```


File: sdm/Win32/unix/time.cpp

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <string.h>
4: #include <time.h>
5: #include <errno.h>
6:
7: #include "WinSock2.h"
8: #include <sys/time.h>
9:
10: #include "sdmLib.h"
11:
12: SDMLIB_API int gettimeofday(struct timeval *tv, struct timezone *tz)
13: {
14:     /*Get time in seconds*/
15:     time_t cur_time;
16:     time(&cur_time);
17:     tv->tv_sec = (long)cur_time;
18:     /*Get milliseconds of current seconds (even though Linux does it in microseconds)*/
19:     SYSTEMTIME sys_time;
20:     GetSystemTime(&sys_time);
21:     tv->tv_usec = sys_time.wMilliseconds;
22:     return 0;
23: }
24:
25: void AlrmHandler(int sig){}
26: void (*sigalrmHandler)(int) = AlrmHandler;
27:
28: typedef struct {
29:     itimerval Value;
30:     int resolutionMs, pendingMode;
31:     MMRESULT TimerId;
32: } RealTimer;
33: RealTimer realTimer = { 0,0,0,0, 10, -1, 0 };
34:
35: SDMLIB_API int getitimer( int which, struct itimerval* value )
36: {
37:     if ( value == NULL )
38:         return EINVAL;
39:     switch ( which )
```

```

40: {
41: case ITIMER_REAL:
42:     *value = realTimer.Value;
43:     break;
44: case ITIMER_VIRTUAL:
45:     break;
46: case ITIMER_PROF:
47:     break;
48: default:
49:     break;
50: }
51: return 0;
52: }
53:
54: void RealTimerHandler(UINT timerId, UINT msg, DWORD_PTR user, DWORD_PTR one,
DWORD_PTR two)
55: {
56: sigalrmHandler(SIGALRM);
57: RealTimer& rt = realTimer;
58: if ( rt.pendingMode == TIME_PERIODIC )
59: {
60:     int intervalMs = rt.Value.it_interval.tv_sec*1000 + rt.Value.it_interval.tv_usec/1000;
61:     realTimer.TimerId = timeSetEvent( intervalMs, realTimer.resolutionMs,
(LPTIMECALLBACK)RealTimerHandler, NULL, realTimer.pendingMode );
62:     rt.pendingMode = -1; // latch this one occurrence
63: }
64: }
65:
66: SDMLIB_API int setitimer( int which, const struct itimerval* value, struct itimerval* old )
67: {
68: int intervalMs, mode;
69: switch ( which )
70: {
71: case ITIMER_REAL:
72:     if ( old != NULL )
73:         *old = realTimer.Value;
74:
75:     realTimer.resolutionMs = value->it_value.tv_usec/1000;
76:     if ( value->it_interval.tv_usec/1000 < realTimer.resolutionMs )
77:         realTimer.resolutionMs = value->it_interval.tv_usec/1000;
78:     realTimer.resolutionMs %= 10;

```

```

79:   if ( realTimer.resolutionMs == 0 ) realTimer.resolutionMs = 10; // if no usec, default to system
clock
80:   timeBeginPeriod(realTimer.resolutionMs);
81:
82:   realTimer.Value = *value;
83:
84:   realTimer.pendingMode = -1;
85:   if ( value->it_interval.tv_sec || value->it_interval.tv_usec )
86:   {
87:       mode = TIME_PERIODIC;
88:       realTimer.pendingMode = TIME_PERIODIC;
89:       intervalMs = value->it_interval.tv_sec*1000+value->it_interval.tv_usec/1000;
90:   }
91:   if ( value->it_value.tv_sec || value->it_value.tv_usec )
92:   {
93:       int periodMs = value->it_value.tv_sec*1000+value->it_value.tv_usec/1000;
94:       if ( periodMs == intervalMs )
95:           realTimer.pendingMode = -1; // leave everything else as PERIODIC
96:       else
97:       {
98:           intervalMs = periodMs;
99:           mode = TIME_ONESHOT;
100:       }
101:   }
102:   realTimer.TimerId      =      timeSetEvent(      intervalMs,      realTimer.resolutionMs,
(LPTIMECALLBACK)RealTimerHandler, NULL, mode );
103:   break;
104: case ITIMER_VIRTUAL:
105: case ITIMER_PROF:
106:     printf("Currently unimplemented \n");
107:     errno = EINVAL;
108:     return -1;
109: default:
110:     errno = EINVAL;
111:     return -1;
112: }
113: return 0;
114: }
115:
116: SDMLIB_API void sigset( int sig, void (*handler)(int) )
117: {

```

```
118:  if ( sig == SIGALRM )
119:      sigalrmHandler = handler;
120: }
121:
122:
```

File: sdm/Win32/unix/byteswap.h

```
1: #ifndef __byteswap_h_
2: #define __byteswap_h_
3:
4: #define bswap_32(x) ( ((0x000000FF&x) << 24) | ((0x0000FF00&x) << 8) | ((0x00FF0000&x) >> 8) |
((0xFF000000&x) >> 24) )
5:
6:
7: #endif // __byteswap_h_
8:
```

File: sdm/Win32/unix/sem.cpp

```
1: #include "sys/sem.h"
2: #include <stdio.h>
3: #define WIN32_LEAN_AND_MEAN
4: #include <windows.h>
5: #include <errno.h>
6: #include <stdarg.h>
7: #include "sdmLib.h"
8:
9:
10: typedef struct sem
11: {
12:     HANDLE handle;
13:     int value, maxVal;
14: } Sem;
15:
16:
17: static Sem* *semaphores = NULL;
18: static int numSemaphores = 1;
19:
20: #define INITIAL_VALUE 0
21: #define MAX_VALUE 1
22:
23: extern int semctl(int semid, int semnum, int cmd, ...)
24: {
25:     if ( semid >= numSemaphores || semnum > 0 )
26:     {
27:         errno = ERANGE;
28:         return -1;
29:     }
30:     Sem* sem = semaphores[semid];
31:     if ( sem == NULL )
32:     {
33:         errno = EINVAL;
34:         return -1;
35:     }
36:     long prevCount = 0;
37:
38:     switch ( cmd )
39:     {
```

```

40: case GETVAL:
41:     printf("semctl(%d,%d,%d, ...) \n",semid, semnum, cmd);
42:     break;
43: case SETVAL://Set the value of semval to arg.val, where arg is the value of the fourth argument to
semctl(). When this command is successfully executed, the semadj value corresponding to the specified
semaphore in all processes is cleared. Requires alter permission, see IPC.
44:     int value;
45:     va_list marker;
46:     va_start( marker, cmd );    /* Initialize variable arguments. */
47:     value = va_arg( marker, int);
48:     va_end( marker );          /* Reset variable arguments.    */
49:     if ( value > sem->value )
50:     {
51:         ReleaseSemaphore( sem->handle, value - sem->value, &prevCount );
52:         sem->value = value;
53:     }
54:     else
55:     {
56:         while ( sem->value > value )
57:         {
58:             WaitForSingleObject( sem->handle, 0 );
59:             sem->value--;
60:         }
61:     }
62:     break;
63: case GETPID://Return the value of sempid. Requires read permission.
64: case GETNCNT://Return the value of semncnt. Requires read permission.
65: case GETZCNT://Return the value of semzcnt. Requires read permission.
66:
67: //The following values of cmd operate on each semval in the set of semaphores:
68: case GETALL://Return the value of semval for each semaphore in the semaphore set and place into
the array pointed to by arg.array, where arg is the fourth argument to semctl(). Requires read permission.
69: case SETALL://Set the value of semval for each semaphore in the semaphore set according to the
array pointed to by arg.array, where arg is the fourth argument to semctl(). When this command is
successfully executed, the semadj values corresponding to each specified semaphore in all processes are
cleared. Requires alter permission.
70:
71: //The following values of cmd are also available:
72: case IPC_STAT://Place the current value of each member of the semid_ds data structure associated
with semid into the structure pointed to by arg.buf, where arg is the fourth argument to semctl(). The
contents of this structure are defined in <sys/sem.h>. Requires read permission.

```

73: case IPC_SET://Set the value of the following members of the semid_ds data structure associated with semid to the corresponding value found in the structure pointed to by arg.buf, where arg is the fourth argument to semctl():

```
74:             // sem_perm.uid
75:             //  sem_perm.gid
76:             //  sem_perm.mode
77:             //  The mode bits specified in IPC are copied into the corresponding bits of the
sem_perm.mode associated with semid. The stored values of any other bits are unspecified. This
command can only be executed by a process that has an effective user ID equal to either that of a process
with appropriate privileges or to the value of sem_perm.cuid or sem_perm.uid in the semid_ds data
structure associated with semid.
```

```
78:     printf("semctl(%d,%d,%d, ...) \n",semid, semnum, cmd);
```

```
79:     break;
```

80: case IPC_RMID://Remove the semaphore-identifier specified by semid from the system and destroy the set of semaphores and semid_ds data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of sem_perm.cuid or sem_perm.uid in the semid_ds data structure associated with semid.

```
81:     CloseHandle(sem->handle);
```

```
82:     free(sem);
```

```
83:     semaphores[semid] = NULL;
```

```
84:     break;
```

```
85: }
```

```
86: return 0;
```

```
87: }
```

```
88:
```

```
89: extern int semget(key_t key, int numSems, unsigned int flags)
```

```
90: {
```

```
91: char name[32];
```

```
92: itoa(key,name,10);
```

```
93: Sem* sem = (Sem*)malloc(sizeof(Sem));
```

```
94: sem->value = INITIAL_VALUE;
```

```
95: sem->maxValue = MAX_VALUE;
```

```
96: sem->handle = CreateSemaphore(NULL, sem->value, sem->maxValue, name);
```

```
97: semaphores = (Sem**)realloc(semaphores,(numSemaphores+1)*sizeof(Sem));
```

```
98: semaphores[numSemaphores] = sem;
```

```
99: return numSemaphores++;
```

```
100: }
```

```
101:
```

```
102: extern int semop(int semid, struct sembuf *sops, size_t nsops)
```

```
103: {
```

```
104:     if ( semid >= numSemaphores || nsops != 1 || sops[0].sem_num >= numSemaphores )
```

```
105:     {
```

```
106:         errno = ERANGE;
```



```

107:     return -1;
108: }
109: Sem* sem = semaphores[semid];
110: if ( sem == NULL )
111: {
112:     errno = EINVAL;
113:     return -1;
114: }
115:
116: int i = 0; // Someday utilize the group of semaphores
117: if ( sops[i].sem_op > 0 )
118: {
119:     long prevCount;
120:     sem->value++;
121:     ReleaseSemaphore( sem->handle, 1, &prevCount );
122: }
123: else if ( sops[i].sem_op < 0 )
124: {
125:     sem->value--;
126:     WaitForSingleObject( sem->handle, INFINITE );
127: }
128: return 0;
129: }
130:

```

File: sdm/Win32/unix/unix/fcntl.h

```
1: #ifndef __unix_fcntl_h_
2: #define __unix_fcntl_h_
3:
4: #include <fcntl.h>
5:
6: // Values for cmd used by fcntl() (the following values are unique) are as follows:
7:
8: #define F_DUPFD      1  // Duplicate file descriptor.
9: #define F_GETFD      2  // Get file descriptor flags.
10: #define F_SETFD      3  // Set file descriptor flags.
11: #define F_GETFL      4  // Get file status flags and file access modes.
12: #define F_SETFL      5  // Set file status flags.
13: #define F_GETLK      6  // Get record locking information.
14: #define F_SETLK      7  // Set record locking information.
15: #define F_SETLKW     8  // Set record locking information; wait if blocked.
16: #define F_GETOWN     9  // Get process or process group ID to receive SIGURG signals.
17: #define F_SETOWN    10  // Set process or process group ID to receive SIGURG signals.
18:
19: // File descriptor flags used for fcntl() are as follows:
20:
21: #define FD_CLOEXEC 11 // Close the file descriptor upon execution of an exec family function.
22:
23: // Values for l_type used for record locking with fcntl() (the following values are unique) are as
follows:
24:
25: #define F_RDLCK     12 // Shared or read lock.
26: #define F_UNLCK     13 // Unlock.
27: #define F_WRLCK     14 // Exclusive or write lock.
28:
29: // The values used for l_whence, SEEK_SET, SEEK_CUR, and SEEK_END shall be defined as
described in <unistd.h>.
30:
31: // The following values are file creation flags and are used in the oflag value to open(). They shall be
bitwise-distinct.
32:
33: // #define O_CREAT      0x0100 // Create file if it does not exist.
34: // #define O_EXCL      0x0400 // Exclusive use flag.
35: #define O_NOCTTY     0x0004 // Do not assign controlling terminal.
36: // #define O_TRUNC      0x0200 // Truncate flag.
37:
```

```

38: // File status flags used for open() and fcntl() are as follows:
39:
40: #define O_APPEND    0x0008 // Set append mode.
41: #define O_DSYNC     0x0040 // [SIO] Write according to synchronized I/O data integrity
    completion.
42: #define O_NONBLOCK   0x0080 // Non-blocking mode.
43: #define O_RSYNC     0x0800 // [SIO] Synchronized read I/O operations.
44: #define O_SYNC      0x1000 // Write according to synchronized I/O file integrity completion.
45:
46: // Mask for use with file access modes is as follows:
47:
48: #define O_ACCMODE 0x0001 // Mask for file access modes.
49:
50: // File access modes used for open() and fcntl() are as follows:
51:
52: #define O_RDONLY 0x0000 // Open for reading only.
53: #define O_RDWR  0x0002 // Open for reading and writing.
54: #define O_WRONLY 0x0001 // Open for writing only.
55: // [XSI] The symbolic names for file modes for use as values of mode_t shall be defined as described
    in <sys/stat.h>.
56:
57: // [ADV] Values for advice used by posix_fadvise() are as follows:
58:
59: #define POSIX_FADV_NORMAL    1 // The application has no advice to give on its behavior
    with respect to the specified data. It is the default characteristic if no advice is given for an open file.
60: #define POSIX_FADV_SEQUENTIAL 2 // The application expects to access the specified data
    sequentially from lower offsets to higher offsets.
61: #define POSIX_FADV_RANDOM    3 // The application expects to access the specified data in
    a random order.
62: #define POSIX_FADV_WILLNEED  4 // The application expects to access the specified
    data in the near future.
63: #define POSIX_FADV_DONTNEED  5 // The application expects that it will not access the
    specified data in the near future.
64: #define POSIX_FADV_NOREUSE   6 // The application expects to access the specified data
    once and then not reuse it thereafter.
65:
66: #endif __unix_fcntl_h_
67:

```

File: sdm/Win32/unix/arpa/inet.h

```
1: #ifndef __arpa_inet_h_
2: #define __arpa_inet_h_
3:
4: #include <unistd.h>
5: // #include <stdio.h>
6:
7: #ifdef _WIN32
8:
9: #ifndef BYTE
10: #define BYTE wBYTE
11: #define CHAR wCHAR
12: #define FLOAT wFLOAT
13: #define LONG wLONG
14: #endif
15:
16: #define WIN32_LEAN_AND_MEAN
17: #include "winsock2.h"
18:
19: #ifdef BYTE
20: #undef BYTE
21: #undef CHAR
22: #undef FLOAT
23: #undef LONG
24: #endif
25:
26: #endif
27:
28: #define SOCK_STREAM 1
29:
30:
31: typedef int socklen_t;
32:
33: extern int setsockopt( int socket, unsigned int level, unsigned int option, void* value, size_t valueSize
);
34:
35: #endif // __arpa_inet_h_
```

File: sdm/Win32/unix/sys/ipc.h

```
1: #ifndef __sys_ipc_h_
2: #define __sys_ipc_h_
3:
4: #define IPC_PRIVATE  0x01
5: #define IPC_CREAT 0x02
6: #define IPC_EXCL  0x04
7: #define IPC_NOWAIT  0x08
8:
9: #endif // __sys_ipc_h_
```

File: sdm/Win32/unix/sys/socket.h

```
1: #ifndef __sys_socket_h_
2: #define __sys_socket_h_
3:
4: #ifdef _WIN32
5:
6: #ifndef BYTE
7: #define BYTE wBYTE
8: #define CHAR wCHAR
9: #define FLOAT wFLOAT
10: #define LONG wLONG
11: #endif
12: #define WIN32_LEAN_AND_MEAN
13: #include "winsock2.h"
14:
15: #ifdef BYTE
16: #undef BYTE
17: #undef CHAR
18: #undef FLOAT
19: #undef LONG
20: #endif
21:
22: #endif
23:
24: extern int setsockopt( int socket, unsigned int level, unsigned int option, void* value, size_t valueSize
);
25:
26: #endif // __sys_socket_h_
```

File: sdm/Win32/unix/sys/time.h

```
1: #ifndef __time_h
2: #define __time_h
3:
4: #include "sdmLib.h"
5:
6: extern SDMLIB_API int gettimeofday(struct timeval *tv, struct timezone *tz);
7:
8: #define WIN32_LEAN_AND_MEAN
9: #include <winsock2.h>
10:
11: struct itimerval {
12: struct timeval it_interval;
13: struct timeval it_value;
14: };
15:
16: #define ITIMER_REAL 0
17: #define ITIMER_VIRTUAL 1
18: #define ITIMER_PROF 2
19:
20: #define SIGALRM 14
21:
22: extern SDMLIB_API void sigset( int sig, void (*handler)(int) );
23: extern SDMLIB_API int setitimer( int which, const struct itimerval* newInterval, struct itimerval*
old );
24: extern SDMLIB_API int getitimer( int which, struct itimerval* old );
25:
26: #endif
```

File: sdm/Win32/unix/sys/poll.h

```
1: #ifndef __sys_poll_h_
2: #define __sys_poll_h_
3:
4: #include "sdmLib.h"
5: #include <winsock2.h>
6:
7: #define POLLIN      0x0001 // Data other than high priority data may be read without blocking.
8:
9: #define POLLOUT      0x0002 // Data may be written without blocking.
10:
11: #define POLLPRI      0x0004 // High priority data may be read without blocking.
12:
13: #define POLLRDNORM    0x0008 // Normal data may be read without blocking.
14:
15: #define POLLRDBAND    0x0010 // Data with a non-zero priority may be read without blocking.
16:
17: #define POLLWRNORM    0x0020 // Normal data may be written without blocking.
18:
19: #define POLLWRBAND    0x0040 // Data with a non-zero priority may be written without blocking.
20:
21: #define POLLERR      0x0080 // An exceptional condition has occurred on the device or socket.
This flag
22:                // is always checked, even if not present in the events bitmask.
23:
24: #define POLLHUP      0x0100 // The device or socket has been disconnected. This flag is always
checked,
25:                // even if not present in the events bitmask. Note that POLLHUP and
POLLOUT
26:                // should never be present in the revents bitmask at the same time. If the
27:                // remote end of a socket is closed, poll() returns a POLLIN event, rather than
28:                // a POLLHUP.
29:
30: #define POLLNVAL      0x0200 // The file descriptor is not open. This flag is always checked, even
if not
31:                // present in the events bitmask.
32: #if(_WIN32_WINNT < 0x0501)
33: struct pollfd
34: {
35: int fd;
36: short events;
```



```
37: short revents;
38: };
39: #else
40:
41: #endif
42: extern int SDMLIB_API poll(struct pollfd* fd, unsigned int resv, int timeout);
43:
44: #endif // __sys_poll_h_
```

File: sdm/Win32/unix/sys/ioctl.h

```
1: #ifndef __sys_ioctl_h_
2: #define __sys_ioctl_h_
3:
4:
5: #endif // __sys_ioctl_h_
```

File: sdm/Win32/unix/sys/sem.h

```
1: #ifndef __sys_sem_h_
2: #define __sys_sem_h_
3:
4: #include <unistd.h>
5: #include <time.h>
6:
7: #define SEM_OP_BLOCK   -1
8: #define SEM_OP_ADD     1
9:
10: struct sembuf
11: {
12:     unsigned short sem_num;
13:     short sem_op;
14:     short sem_flg;
15: };
16: union semun
17: {
18:     int val;           // value for SETVAL
19:     struct semid_ds *buf; // buffer for IPC_STAT & IPC_SET
20:     unsigned short *array; // array for GETALL & SETALL
21:     struct seminfo *__buf; // buffer for IPC_INFO
22:     void *__pad;
23: };
24: struct ipc_perm {
25:     unsigned short user;
26:     unsigned short group;
27:     unsigned short other;
28: };
29:
30: struct semid_ds {
31:     struct ipc_perm sem_perm;    /* permissions .. see ipc.h */
32:     time_t sem_otime;    /* last semop time */
33:     time_t sem_ctime;    /* last change time */
34:     struct sem *sem_base;    /* ptr to first semaphore in array */
35:     struct wait_queue *eventn;
36:     struct wait_queue *eventz;
37:     struct sem_undo *undo;    /* undo requests on this array */
38:     ushort sem_nsems;    /* no. of semaphores in array */
39: };
```

```

40:
41: // Cmd values
42: #define GETVAL 1 //Return the value of semval, see <sys/sem.h>. Requires read permission.
43: #define SETVAL 2 //Set the value of semval to arg.val, where arg is the value of the fourth
argument to semctl(). When this command is successfully executed, the semadj value corresponding to
the specified semaphore in all processes is cleared. Requires alter permission, see IPC.
44: #define GETPID 3 //Return the value of sempid. Requires read permission.
45: #define GETNCNT 4 //Return the value of semncnt. Requires read permission.
46: #define GETZCNT 5 //Return the value of semzcnt. Requires read permission.
47:
48: //The following values of cmd operate on each semval in the set of semaphores:
49: #define GETALL 6 //Return the value of semval for each semaphore in the semaphore set and place
into the array pointed to by arg.array, where arg is the fourth argument to semctl(). Requires read
permission.
50: #define SETALL 7 //Set the value of semval for each semaphore in the semaphore set according to
the array pointed to by arg.array, where arg is the fourth argument to semctl(). When this command is
successfully executed, the semadj values corresponding to each specified semaphore in all processes are
cleared. Requires alter permission.
51:
52: //The following values of cmd are also available:
53: #define IPC_STAT 8 //Place the current value of each member of the semid_ds data structure
associated with semid into the structure pointed to by arg.buf, where arg is the fourth argument to
semctl(). The contents of this structure are defined in <sys/sem.h>. Requires read permission.
54: #define IPC_SET 9 //Set the value of the following members of the semid_ds data structure
associated with semid to the corresponding value found in the structure pointed to by arg.buf, where arg is
the fourth argument to semctl():
55: //  sem_perm.uid
56: //  sem_perm.gid
57: //  sem_perm.mode
58: //  The mode bits specified in IPC are copied into the corresponding bits of the sem_perm.mode
associated with semid. The stored values of any other bits are unspecified. This command can only be
executed by a process that has an effective user ID equal to either that of a process with appropriate
privileges or to the value of sem_perm.cuid or sem_perm.uid in the semid_ds data structure associated
with semid.
59: #define IPC_RMID 10 //Remove the semaphore-identifier specified by semid from the system and
destroy the set of semaphores and semid_ds data structure associated with it. This command can only be
executed by a process that has an effective user ID equal to either that of a process with appropriate
privileges or to the value of sem_perm.cuid or sem_perm.uid in the semid_ds data structure associated
with semid.
60:
61: extern int semctl(int semid, int semnum, int cmd, ...);
62: extern int semget(key_t key, int numSems, unsigned int flags);
63: extern int semop(int semid, struct sembuf *sops, size_t nsops);
64:
65: #endif // __sys_sem_h_

```

File: sdm/Win32/unix/sys/wait.h

```
1: #ifndef __sys_wait_h_
2: #define __sys_wait_h_
3:
4: #include <unistd.h>
5:
6: #define SIGCHLD 33
7: #include <signal.h>
8: // The siginfo_t type shall be defined as described in <signal.h> .
9: union sigval {
10: int sival_int;
11: void *sival_ptr;
12: };
13:
14: typedef struct {
15: int si_signo;
16: int si_code;
17: union sigval si_value;
18: int si_errno;
19: pid_t si_pid;
20: uid_t si_uid;
21: void *si_addr;
22: int si_status;
23: int si_band;
24: } siginfo_t;
25:
26:
27: // #include <sys/resource.h>
28: // The rusage structure shall be defined as described in .
29: #ifdef _WIN32
30:
31: #ifndef BYTE
32: #define BYTE wBYTE
33: #define CHAR wCHAR
34: #define FLOAT wFLOAT
35: #define LONG wLONG
36: #endif
37:
38: #define WIN32_LEAN_AND_MEAN
39: #include "winsock2.h"
```

```

40:
41: #ifdef BYTE
42: #undef BYTE
43: #undef CHAR
44: #undef FLOAT
45: #undef LONG
46: #endif
47:
48: #endif
49:
50: struct rusage
51: {
52: struct timeval ru_utime;    /* user time used */
53: struct timeval ru_stime;    /* system time used */
54: };
55:
56: // #define WNOHANG          1 // Do not hang if no status is available; return immediately.
57: #define WUNTRACED          2 // Report status of stopped child process.
58: #define WEXITSTATUS        3 // Return exit status.
59: #define WIFCONTINUED        4 // True if child has been continued.
60: #define WIFEXITED           5 // True if child exited normally.
61: #define WIFSIGNALED         6 // True if child exited due to uncaught signal.
62: #define WIFSTOPPED          7 // True if child is currently stopped.
63: #define WSTOPSIG            8 // Return signal number that caused process to stop.
64: #define WTERMSIG            9 // Return signal number that caused process to terminate.
65: #define WEXITED             10 // Wait for processes that have exited.
66: #define WSTOPPED            11 // Status is returned for any child that has stopped upon receipt of a
    signal.
67: #define WCONTINUED          12 // Status is returned for any child that was stopped and has been
    continued.
68: #define WNOHANG             13 // Return immediately if there are no children to wait for.
69: #define WNOWAIT             14 // Keep the process whose status is returned in infop in a waitable
    state.
70:
71: typedef enum { P_ALL, P_PID, P_PGID } idtype_t;
72: typedef unsigned int id_t;
73:
74: extern pid_t wait(int * status);
75:
76: extern int waitid(idtype_t idType, id_t id, siginfo_t * info, int resv);
77:
78: extern pid_t waitpid(pid_t pid, int * resv1, int resv2);

```

79:

80: #endif // __sys_wait_h__

File: sdm/Win32/unix/netinet/in.h

```
1: #ifndef __netinet_in_h_
2: #define __netinet_in_h_
3:
4: #ifdef _WIN32
5:
6: #ifndef BYTE
7: #define BYTE wBYTE
8: #define CHAR wCHAR
9: #define FLOAT wFLOAT
10: #define LONG wLONG
11: #endif
12:
13: #define WIN32_LEAN_AND_MEAN
14: #include "winsock2.h"
15:
16: #ifdef BYTE
17: #undef BYTE
18: #undef CHAR
19: #undef FLOAT
20: #undef LONG
21: #endif
22:
23: #endif
24:
25: typedef unsigned int in_addr_t;
26:
27: #endif // __netinet_in_h_
```


File: sdm/Win32/Regex/regex.h

```
1: /* Definitions for data structures and routines for the regular
2:  expression library.
3:  Copyright (C) 1985,1989-93,1995-98,2000,2001,2002,2003,2005,2006
4:  Free Software Foundation, Inc.
5:  This file is part of the GNU C Library.
6:
7:  The GNU C Library is free software; you can redistribute it and/or
8:  modify it under the terms of the GNU Lesser General Public
9:  License as published by the Free Software Foundation; either
10:  version 2.1 of the License, or (at your option) any later version.
11:
12:  The GNU C Library is distributed in the hope that it will be useful,
13:  but WITHOUT ANY WARRANTY; without even the implied warranty of
14:  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
15:  Lesser General Public License for more details.
16:
17:  You should have received a copy of the GNU Lesser General Public
18:  License along with the GNU C Library; if not, write to the Free
19:  Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
20:  02111-1307 USA.  */
21:
22: #ifndef _REGEX_H
23: #define _REGEX_H 1
24:
25: #include <sys/types.h>
26:
27: #ifndef __GNUC__
28: # define __DLL_IMPORT__  __declspec(dllimport)
29: # define __DLL_EXPORT__  __declspec(dllexport)
30: #else
31: # define __DLL_IMPORT__  __attribute__((dllimport)) extern
32: # define __DLL_EXPORT__  __attribute__((dllexport)) extern
33: #endif
34:
35: #if (defined __WIN32__) || (defined _WIN32)
36: # ifdef BUILD_REGEX_DLL
37: #  define REGEX_DLL_IMPEXP  __DLL_EXPORT__
38: # elif defined(REGEX_STATIC)
39: #  define REGEX_DLL_IMPEXP
```

```

40: # elif defined (USE_REGEX_DLL)
41: # define REGEX_DLL_IMPEXP __DLL_IMPORT__
42: # elif defined (USE_REGEX_STATIC)
43: # define REGEX_DLL_IMPEXP
44: # else /* assume USE_REGEX_DLL */
45: # define REGEX_DLL_IMPEXP __DLL_IMPORT__
46: # endif
47: #else /* __WIN32__ */
48: # define REGEX_DLL_IMPEXP
49: #endif
50:
51: /* Allow the use in C++ code. */
52: #ifdef __cplusplus
53: extern "C" {
54: #endif
55:
56: /* The following two types have to be signed and unsigned integer type
57:  wide enough to hold a value of a pointer. For most ANSI compilers
58:  ptrdiff_t and size_t should be likely OK. Still size of these two
59:  types is 2 for Microsoft C. Ugh... */
60: typedef long int s_reg_t;
61: typedef unsigned long int active_reg_t;
62:
63: /* The following bits are used to determine the regexp syntax we
64:  recognize. The set/not-set meanings are chosen so that Emacs syntax
65:  remains the value 0. The bits are given in alphabetical order, and
66:  the definitions shifted by one from the previous bit; thus, when we
67:  add or remove a bit, only one other definition need change. */
68: typedef unsigned long int reg_syntax_t;
69:
70: /* If this bit is not set, then \ inside a bracket expression is literal.
71:  If set, then such a \ quotes the following character. */
72: #define RE_BACKSLASH_ESCAPE_IN_LISTS ((unsigned long int) 1)
73:
74: /* If this bit is not set, then + and ? are operators, and \+ and \? are
75:  literals.
76:  If set, then \+ and \? are operators and + and ? are literals. */
77: #define RE_BK_PLUS_QM (RE_BACKSLASH_ESCAPE_IN_LISTS << 1)
78:
79: /* If this bit is set, then character classes are supported. They are:
80:  [:alpha:], [:upper:], [:lower:], [:digit:], [:alnum:], [:xdigit:],

```

```

81:  [:space:], [:print:], [:punct:], [:graph:], and [:cntrl:].
82:  If not set, then character classes are not supported. */
83: #define RE_CHAR_CLASSES (RE_BK_PLUS_QM << 1)
84:
85: /* If this bit is set, then ^ and $ are always anchors (outside bracket
86:    expressions, of course).
87:  If this bit is not set, then it depends:
88:    ^ is an anchor if it is at the beginning of a regular
89:    expression or after an open-group or an alternation operator;
90:    $ is an anchor if it is at the end of a regular expression, or
91:    before a close-group or an alternation operator.
92:
93:  This bit could be (re)combined with RE_CONTEXT_INDEP_OPS, because
94:  POSIX draft 11.2 says that * etc. in leading positions is undefined.
95:  We already implemented a previous draft which made those constructs
96:  invalid, though, so we haven't changed the code back. */
97: #define RE_CONTEXT_INDEP_ANCHORS (RE_CHAR_CLASSES << 1)
98:
99: /* If this bit is set, then special characters are always special
100:    regardless of where they are in the pattern.
101:  If this bit is not set, then special characters are special only in
102:    some contexts; otherwise they are ordinary. Specifically,
103:    * + ? and intervals are only special when not after the beginning,
104:    open-group, or alternation operator. */
105: #define RE_CONTEXT_INDEP_OPS (RE_CONTEXT_INDEP_ANCHORS << 1)
106:
107: /* If this bit is set, then *, +, ?, and { cannot be first in an re or
108:    immediately after an alternation or begin-group operator. */
109: #define RE_CONTEXT_INVALID_OPS (RE_CONTEXT_INDEP_OPS << 1)
110:
111: /* If this bit is set, then . matches newline.
112:  If not set, then it doesn't. */
113: #define RE_DOT_NEWLINE (RE_CONTEXT_INVALID_OPS << 1)
114:
115: /* If this bit is set, then . doesn't match NUL.
116:  If not set, then it does. */
117: #define RE_DOT_NOT_NULL (RE_DOT_NEWLINE << 1)
118:
119: /* If this bit is set, nonmatching lists [^...] do not match newline.
120:  If not set, they do. */
121: #define RE_HAT_LISTS_NOT_NEWLINE (RE_DOT_NOT_NULL << 1)

```

```

122:
123: /* If this bit is set, either \{... \} or {...} defines an
124:    interval, depending on RE_NO_BK_BRACES.
125:    If not set, \{, \}, {, and } are literals. */
126: #define RE_INTERVALS (RE_HAT_LISTS_NOT_NEWLINE << 1)
127:
128: /* If this bit is set, +, ? and | aren't recognized as operators.
129:    If not set, they are. */
130: #define RE_LIMITED_OPS (RE_INTERVALS << 1)
131:
132: /* If this bit is set, newline is an alternation operator.
133:    If not set, newline is literal. */
134: #define RE_NEWLINE_ALT (RE_LIMITED_OPS << 1)
135:
136: /* If this bit is set, then `{...}' defines an interval, and \{ and \}
137:    are literals.
138:    If not set, then `{... \}' defines an interval. */
139: #define RE_NO_BK_BRACES (RE_NEWLINE_ALT << 1)
140:
141: /* If this bit is set, (...) defines a group, and \( and \) are literals.
142:    If not set, \(... \) defines a group, and ( and ) are literals. */
143: #define RE_NO_BK_PARENS (RE_NO_BK_BRACES << 1)
144:
145: /* If this bit is set, then \<digit> matches <digit>.
146:    If not set, then \<digit> is a back-reference. */
147: #define RE_NO_BK_REFS (RE_NO_BK_PARENS << 1)
148:
149: /* If this bit is set, then | is an alternation operator, and \| is literal.
150:    If not set, then \| is an alternation operator, and | is literal. */
151: #define RE_NO_BK_VBAR (RE_NO_BK_REFS << 1)
152:
153: /* If this bit is set, then an ending range point collating higher
154:    than the starting range point, as in [z-a], is invalid.
155:    If not set, then when ending range point collates higher than the
156:    starting range point, the range is ignored. */
157: #define RE_NO_EMPTY_RANGES (RE_NO_BK_VBAR << 1)
158:
159: /* If this bit is set, then an unmatched ) is ordinary.
160:    If not set, then an unmatched ) is invalid. */
161: #define RE_UNMATCHED_RIGHT_PAREN_ORD (RE_NO_EMPTY_RANGES << 1)
162:

```

```

163: /* If this bit is set, succeed as soon as we match the whole pattern,
164:  without further backtracking. */
165: #define RE_NO_POSIX_BACKTRACKING (RE_UNMATCHED_RIGHT_PAREN_ORD << 1)
166:
167: /* If this bit is set, do not process the GNU regex operators.
168:  If not set, then the GNU regex operators are recognized. */
169: #define RE_NO_GNU_OPS (RE_NO_POSIX_BACKTRACKING << 1)
170:
171: /* If this bit is set, turn on internal regex debugging.
172:  If not set, and debugging was on, turn it off.
173:  This only works if regex.c is compiled -DDEBUG.
174:  We define this bit always, so that all that's needed to turn on
175:  debugging is to recompile regex.c; the calling code can always have
176:  this bit set, and it won't affect anything in the normal case. */
177: #define RE_DEBUG (RE_NO_GNU_OPS << 1)
178:
179: /* If this bit is set, a syntactically invalid interval is treated as
180:  a string of ordinary characters. For example, the ERE 'a{1}' is
181:  treated as 'a \{1'. */
182: #define RE_INVALID_INTERVAL_ORD (RE_DEBUG << 1)
183:
184: /* If this bit is set, then ignore case when matching.
185:  If not set, then case is significant. */
186: #define RE_ICASE (RE_INVALID_INTERVAL_ORD << 1)
187:
188: /* This bit is used internally like RE_CONTEXT_INDEP_ANCHORS but only
189:  for ^, because it is difficult to scan the regex backwards to find
190:  whether ^ should be special. */
191: #define RE_CARET_ANCHORS_HERE (RE_ICASE << 1)
192:
193: /* If this bit is set, then \{ cannot be first in an bre or
194:  immediately after an alternation or begin-group operator. */
195: #define RE_CONTEXT_INVALID_DUP (RE_CARET_ANCHORS_HERE << 1)
196:
197: /* If this bit is set, then no_sub will be set to 1 during
198:  re_compile_pattern. */
199: #define RE_NO_SUB (RE_CONTEXT_INVALID_DUP << 1)
200:
201: /* This global variable defines the particular regexp syntax to use (for
202:  some interfaces). When a regexp is compiled, the syntax used is
203:  stored in the pattern buffer, so changing this does not affect

```

```

204: already-compiled regexps. */
205: REGEX_DLL_IMPEXP reg_syntax_t re_syntax_options;
206:
207: /* Define combinations of the above bits for the standard possibilities.
208:  (The [[[ comments delimit what gets put into the Texinfo file, so
209:  don't delete them!) */
210: /* [[[begin syntaxes]]] */
211: #define RE_SYNTAX_EMACS 0
212:
213: #define RE_SYNTAX_AWK \
214: (RE_BACKSLASH_ESCAPE_IN_LISTS | RE_DOT_NOT_NULL \
215: | RE_NO_BK_PARENS | RE_NO_BK_REFS \
216: | RE_NO_BK_VBAR | RE_NO_EMPTY_RANGES \
217: | RE_DOT_NEWLINE | RE_CONTEXT_INDEP_ANCHORS \
218: | RE_UNMATCHED_RIGHT_PAREN_ORD | RE_NO_GNU_OPS)
219:
220: #define RE_SYNTAX_GNU_AWK \
221: ((RE_SYNTAX_POSIX_EXTENDED | RE_BACKSLASH_ESCAPE_IN_LISTS | RE_DEBUG) \
222: & ~(RE_DOT_NOT_NULL | RE_INTERVALS | RE_CONTEXT_INDEP_OPS \
223: | RE_CONTEXT_INVALID_OPS ))
224:
225: #define RE_SYNTAX_POSIX_AWK \
226: (RE_SYNTAX_POSIX_EXTENDED | RE_BACKSLASH_ESCAPE_IN_LISTS \
227: | RE_INTERVALS | RE_NO_GNU_OPS)
228:
229: #define RE_SYNTAX_GREP \
230: (RE_BK_PLUS_QM | RE_CHAR_CLASSES \
231: | RE_HAT_LISTS_NOT_NEWLINE | RE_INTERVALS \
232: | RE_NEWLINE_ALT)
233:
234: #define RE_SYNTAX_EGREP \
235: (RE_CHAR_CLASSES | RE_CONTEXT_INDEP_ANCHORS \
236: | RE_CONTEXT_INDEP_OPS | RE_HAT_LISTS_NOT_NEWLINE \
237: | RE_NEWLINE_ALT | RE_NO_BK_PARENS \
238: | RE_NO_BK_VBAR)
239:
240: #define RE_SYNTAX_POSIX_EGREP \
241: (RE_SYNTAX_EGREP | RE_INTERVALS | RE_NO_BK_BRACES \
242: | RE_INVALID_INTERVAL_ORD)
243:

```

```

244: /* P1003.2/D11.2, section 4.20.7.1, lines 5078ff. */
245: #define RE_SYNTAX_ED RE_SYNTAX_POSIX_BASIC
246:
247: #define RE_SYNTAX_SED RE_SYNTAX_POSIX_BASIC
248:
249: /* Syntax bits common to both basic and extended POSIX regex syntax. */
250: #define _RE_SYNTAX_POSIX_COMMON \
251: (RE_CHAR_CLASSES | RE_DOT_NEWLINE | RE_DOT_NOT_NULL \
252: | RE_INTERVALS | RE_NO_EMPTY_RANGES)
253:
254: #define RE_SYNTAX_POSIX_BASIC \
255: (_RE_SYNTAX_POSIX_COMMON | RE_BK_PLUS_QM | RE_CONTEXT_INVALID_DUP)
256:
257: /* Differs from ..._POSIX_BASIC only in that RE_BK_PLUS_QM becomes
258: RE_LIMITED_OPS, i.e., \? \+ \| are not recognized. Actually, this
259: isn't minimal, since other operators, such as \`, aren't disabled. */
260: #define RE_SYNTAX_POSIX_MINIMAL_BASIC \
261: (_RE_SYNTAX_POSIX_COMMON | RE_LIMITED_OPS)
262:
263: #define RE_SYNTAX_POSIX_EXTENDED \
264: (_RE_SYNTAX_POSIX_COMMON | RE_CONTEXT_INDEP_ANCHORS \
265: | RE_CONTEXT_INDEP_OPS | RE_NO_BK_BRACES \
266: | RE_NO_BK_PARENS | RE_NO_BK_VBAR \
267: | RE_CONTEXT_INVALID_OPS | RE_UNMATCHED_RIGHT_PAREN_ORD)
268:
269: /* Differs from ..._POSIX_EXTENDED in that RE_CONTEXT_INDEP_OPS is
270: removed and RE_NO_BK_REFS is added. */
271: #define RE_SYNTAX_POSIX_MINIMAL_EXTENDED \
272: (_RE_SYNTAX_POSIX_COMMON | RE_CONTEXT_INDEP_ANCHORS \
273: | RE_CONTEXT_INVALID_OPS | RE_NO_BK_BRACES \
274: | RE_NO_BK_PARENS | RE_NO_BK_REFS \
275: | RE_NO_BK_VBAR | RE_UNMATCHED_RIGHT_PAREN_ORD)
276: /* [[[end syntaxes]]] */
277:
278: /* Maximum number of duplicates an interval can allow. Some systems
279: (erroneously) define this in other header files, but we want our
280: value, so remove any previous define. */
281: #ifdef RE_DUP_MAX
282: #undef RE_DUP_MAX
283: #endif
284: /* If sizeof(int) == 2, then ((1 << 15) - 1) overflows. */

```

```

285: #define RE_DUP_MAX (0x7fff)
286:
287:
288: /* POSIX `cflags' bits (i.e., information for `regcomp'). */
289:
290: /* If this bit is set, then use extended regular expression syntax.
291:  If not set, then use basic regular expression syntax. */
292: #define REG_EXTENDED 1
293:
294: /* If this bit is set, then ignore case when matching.
295:  If not set, then case is significant. */
296: #define REG_ICASE (REG_EXTENDED << 1)
297:
298: /* If this bit is set, then anchors do not match at newline
299:  characters in the string.
300:  If not set, then anchors do match at newlines. */
301: #define REG_NEWLINE (REG_ICASE << 1)
302:
303: /* If this bit is set, then report only success or fail in regexec.
304:  If not set, then returns differ between not matching and errors. */
305: #define REG_NOSUB (REG_NEWLINE << 1)
306:
307:
308: /* POSIX `eflags' bits (i.e., information for regexec). */
309:
310: /* If this bit is set, then the beginning-of-line operator doesn't match
311:  the beginning of the string (presumably because it's not the
312:  beginning of a line).
313:  If not set, then the beginning-of-line operator does match the
314:  beginning of the string. */
315: #define REG_NOTBOL 1
316:
317: /* Like REG_NOTBOL, except for the end-of-line. */
318: #define REG_NOTEOL (1 << 1)
319:
320: /* Use PMATCH[0] to delimit the start and end of the search in the
321:  buffer. */
322: #define REG_STARTEND (1 << 2)
323:
324:
325: /* If any error codes are removed, changed, or added, update the

```



```

326: `re_error_msg' table in regex.c. */
327: typedef enum
328: {
329: #ifdef _XOPEN_SOURCE
330: REG_ENOSYS = -1, /* This will never happen for this implementation. */
331: #endif
332:
333: REG_NOERROR = 0, /* Success. */
334: REG_NOMATCH, /* Didn't find a match (for regex). */
335:
336: /* POSIX regcomp return error codes. (In the order listed in the
337:  standard.) */
338: REG_BADPAT, /* Invalid pattern. */
339: REG_ECOLLATE, /* Invalid collating element. */
340: REG_ECTYPE, /* Invalid character class name. */
341: REG_EESCAPE, /* Trailing backslash. */
342: REG_ESUBREG, /* Invalid back reference. */
343: REG_EBRACK, /* Unmatched left bracket. */
344: REG_EPAREN, /* Parenthesis imbalance. */
345: REG_EBRACE, /* Unmatched \{. */
346: REG_BADBR, /* Invalid contents of \{ \}. */
347: REG_ERANGE, /* Invalid range end. */
348: REG_ESPACE, /* Ran out of memory. */
349: REG_BADRPT, /* No preceding re for repetition op. */
350:
351: /* Error codes we've added. */
352: REG_EEND, /* Premature end. */
353: REG_ESIZE, /* Compiled pattern bigger than 2^16 bytes. */
354: REG_ERPAREN /* Unmatched ) or \); not returned from regcomp. */
355: } reg_errcode_t;
356:
357: /* This data structure represents a compiled pattern. Before calling
358:  the pattern compiler, the fields `buffer', `allocated', `fastmap',
359:  `translate', and `no_sub' can be set. After the pattern has been
360:  compiled, the `re_nsub' field is available. All other fields are
361:  private to the regex routines. */
362:
363: #ifndef RE_TRANSLATE_TYPE
364: # define RE_TRANSLATE_TYPE unsigned char *
365: #endif
366:

```

```

367: struct re_pattern_buffer
368: {
369:     /* Space that holds the compiled pattern. It is declared as
370:        `unsigned char *' because its elements are sometimes used as
371:        array indexes. */
372:     unsigned char *buffer;
373:
374:     /* Number of bytes to which `buffer' points. */
375:     unsigned long int allocated;
376:
377:     /* Number of bytes actually used in `buffer'. */
378:     unsigned long int used;
379:
380:     /* Syntax setting with which the pattern was compiled. */
381:     reg_syntax_t syntax;
382:
383:     /* Pointer to a fastmap, if any, otherwise zero. re_search uses the
384:        fastmap, if there is one, to skip over impossible starting points
385:        for matches. */
386:     char *fastmap;
387:
388:     /* Either a translate table to apply to all characters before
389:        comparing them, or zero for no translation. The translation is
390:        applied to a pattern when it is compiled and to a string when it
391:        is matched. */
392:     RE_TRANSLATE_TYPE translate;
393:
394:     /* Number of subexpressions found by the compiler. */
395:     size_t re_nsub;
396:
397:     /* Zero if this pattern cannot match the empty string, one else.
398:        Well, in truth it's used only in `re_search_2', to see whether or
399:        not we should use the fastmap, so we don't set this absolutely
400:        perfectly; see `re_compile_fastmap' (the `duplicate' case). */
401:     unsigned can_be_null : 1;
402:
403:     /* If REGS_UNALLOCATED, allocate space in the `regs' structure
404:        for `max (RE_NREGS, re_nsub + 1)' groups.
405:        If REGS_REALLOCATE, reallocate space if necessary.
406:        If REGS_FIXED, use what's there. */
407: #define REGS_UNALLOCATED 0

```

```

408: #define REGS_REALLOCATE 1
409: #define REGS_FIXED 2
410: unsigned regs_allocated : 2;
411:
412: /* Set to zero when `regex_compile' compiles a pattern; set to one
413:    by `re_compile_fastmap' if it updates the fastmap. */
414: unsigned fastmap_accurate : 1;
415:
416: /* If set, `re_match_2' does not return information about
417:    subexpressions. */
418: unsigned no_sub : 1;
419:
420: /* If set, a beginning-of-line anchor doesn't match at the beginning
421:    of the string. */
422: unsigned not_bol : 1;
423:
424: /* Similarly for an end-of-line anchor. */
425: unsigned not_eol : 1;
426:
427: /* If true, an anchor at a newline matches. */
428: unsigned newline_anchor : 1;
429: };
430:
431: typedef struct re_pattern_buffer regex_t;
432:
433: /* Type for byte offsets within the string.  POSIX mandates this. */
434: typedef int regoff_t;
435:
436:
437: /* This is the structure we store register match data in.  See
438:    regex.texinfo for a full description of what registers match. */
439: struct re_registers
440: {
441:   unsigned num_regs;
442:   regoff_t *start;
443:   regoff_t *end;
444: };
445:
446:
447: /* If `regs_allocated' is REGS_UNALLOCATED in the pattern buffer,
448:    `re_match_2' returns information about at least this many registers

```

```

449:  the first time a `regs' structure is passed. */
450: #ifndef RE_NREGS
451: # define RE_NREGS 30
452: #endif
453:
454:
455: /* POSIX specification for registers.  Aside from the different names than
456:  `re_registers', POSIX uses an array of structures, instead of a
457:  structure of arrays. */
458: typedef struct
459: {
460:  regoff_t rm_so; /* Byte offset from string's start to substring's start. */
461:  regoff_t rm_eo; /* Byte offset from string's start to substring's end. */
462: } regmatch_t;
463:
464: /* Declarations for routines. */
465:
466: /* Sets the current default syntax to SYNTAX, and return the old syntax.
467:  You can also simply assign to the `re_syntax_options' variable. */
468: REGEX_DLL_IMPEXP reg_syntax_t re_set_syntax (reg_syntax_t __syntax);
469:
470: /* Compile the regular expression PATTERN, with length LENGTH
471:  and syntax given by the global `re_syntax_options', into the buffer
472:  BUFFER. Return NULL if successful, and an error string if not. */
473: REGEX_DLL_IMPEXP const char *re_compile_pattern (const char *__pattern, size_t __length,
474:                                                  struct re_pattern_buffer *__buffer);
475:
476:
477: /* Compile a fastmap for the compiled pattern in BUFFER; used to
478:  accelerate searches. Return 0 if successful and -2 if was an
479:  internal error. */
480: REGEX_DLL_IMPEXP int re_compile_fastmap (struct re_pattern_buffer *__buffer);
481:
482:
483: /* Search in the string STRING (with length LENGTH) for the pattern
484:  compiled into BUFFER. Start searching at position START, for RANGE
485:  characters. Return the starting position of the match, -1 for no
486:  match, or -2 for an internal error. Also return register
487:  information in REGS (if REGS and BUFFER->no_sub are nonzero). */
488: REGEX_DLL_IMPEXP int re_search (struct re_pattern_buffer *__buffer, const char *__string,
489:                                int __length, int __start, int __range,

```

```

490:         struct re_registers *__regs);
491:
492:
493: /* Like `re_search', but search in the concatenation of STRING1 and
494:  STRING2. Also, stop searching at index START + STOP. */
495: REGEX_DLL_IMPEXP int re_search_2 (struct re_pattern_buffer *__buffer,
496:         const char *__string1, int __length1,
497:         const char *__string2, int __length2, int __start,
498:         int __range, struct re_registers *__regs, int __stop);
499:
500:
501: /* Like `re_search', but return how many characters in STRING the regexp
502:  in BUFFER matched, starting at position START. */
503: REGEX_DLL_IMPEXP int re_match (struct re_pattern_buffer *__buffer, const char *__string,
504:         int __length, int __start, struct re_registers *__regs);
505:
506:
507: /* Relates to `re_match' as `re_search_2' relates to `re_search'. */
508: REGEX_DLL_IMPEXP int re_match_2 (struct re_pattern_buffer *__buffer,
509:         const char *__string1, int __length1,
510:         const char *__string2, int __length2, int __start,
511:         struct re_registers *__regs, int __stop);
512:
513:
514: /* Set REGS to hold NUM_REGS registers, storing them in STARTS and
515:  ENDS. Subsequent matches using BUFFER and REGS will use this memory
516:  for recording register information. STARTS and ENDS must be
517:  allocated with malloc, and must each be at least `NUM_REGS * sizeof
518:  (regoff_t)' bytes long.
519:
520:  If NUM_REGS == 0, then subsequent matches should allocate their own
521:  register data.
522:
523:  Unless this function is called, the first search or match using
524:  PATTERN_BUFFER will allocate its own register data, without
525:  freeing the old data. */
526: REGEX_DLL_IMPEXP void re_set_registers (struct re_pattern_buffer *__buffer,
527:         struct re_registers *__regs,
528:         unsigned int __num_regs,
529:         regoff_t *__starts, regoff_t *__ends);
530:

```

```

531: #if defined _REGEX_RE_COMP || defined _LIBC
532: # ifndef _CRAY
533: /* 4.2 bsd compatibility. */
534: REGEX_DLL_IMPEXP char *re_comp (const char *);
535: REGEX_DLL_IMPEXP int re_exec (const char *);
536: # endif
537: #endif
538:
539: /* GCC 2.95 and later have "__restrict"; C99 compilers have
540:  "restrict", and "configure" may have defined "restrict". */
541: #ifndef __restrict
542: # if ! (2 < __GNUC__ || (2 == __GNUC__ && 95 <= __GNUC_MINOR__))
543: #  if defined restrict || 199901L <= __STDC_VERSION__
544: #   define __restrict restrict
545: #  else
546: #   define __restrict
547: #  endif
548: # endif
549: #endif
550: /* gcc 3.1 and up support the [restrict] syntax. */
551: #ifndef __restrict_arr
552: # if (__GNUC__ > 3 || (__GNUC__ == 3 && __GNUC_MINOR__ >= 1)) \
553:   && !defined __GNUG__
554: #  define __restrict_arr __restrict
555: # else
556: #  define __restrict_arr
557: # endif
558: #endif
559:
560: /* POSIX compatibility. */
561: REGEX_DLL_IMPEXP int regcomp (regex_t *__restrict __preg,
562:      const char *__restrict __pattern,
563:      int __cflags);
564:
565: REGEX_DLL_IMPEXP int regexec (const regex_t *__restrict __preg,
566:      const char *__restrict __string, size_t __nmatch,
567:      regmatch_t __pmatch[__restrict_arr],
568:      int __eflags);
569:
570: REGEX_DLL_IMPEXP size_t regerror (int __errcode, const regex_t *__restrict __preg,
571:      char *__restrict __errbuf, size_t __errbuf_size);

```

```
572:
573: REGEX_DLL_IMPEXP void regfree (regex_t *__preg);
574:
575:
576: #ifdef __cplusplus
577: }
578: #endif /* C++ */
579:
580: #endif /* regex.h */
```

File: sdm/Win32/sdmLib/sdmLib.cpp

```
1: // sdmLib.cpp : Defines the entry point for the DLL application.
2: //
3:
4: #include <winsock2.h>
5: #include "sdmLib.h"
6: BOOL APIENTRY DllMain( HANDLE hModule,
7:         DWORD ul_reason_for_call,
8:         LPVOID lpReserved
9:         )
10: {
11:     unsigned short wVersionRequested;
12:     WSADATA wsaData;
13:     int err;
14:     switch (ul_reason_for_call)
15:     {
16:     case DLL_PROCESS_ATTACH:
17:         wVersionRequested = MAKEWORD( 2, 2 );
18:         err = WSAStartup( wVersionRequested, &wsaData );
19:         if ( err != 0 ) {
20:             // Tell the user that we could not find a usable WinSock DLL.
21:             return -1;
22:         }
23:         break;
24:     case DLL_THREAD_ATTACH:
25:         break;
26:     case DLL_THREAD_DETACH:
27:         break;
28:     case DLL_PROCESS_DETACH:
29:         WSACleanup();
30:         break;
31:     }
32:     return TRUE;
33: }
34:
35: // This is an example of an exported variable
36: SDMLIB_API int nsdmLib=0;
37:
38: // This is an example of an exported function.
39: SDMLIB_API int fnsdmLib(void)
```



```
40: {  
41: return 42;  
42: }  
43:  
44: // This is the constructor of a class that has been exported.  
45: // see sdmLib.h for the class definition  
46: //CsdmLib::CsdmLib()  
47: //{  
48: //    return;  
49: //}
```

BIBLIOGRAPHY

1. Arney, D., Wilhite, A., "A Modeling Environment for the Optimization of Space Architectures," Paper No. AIAA-2010-8665, *AIAA SPACE 2010 Conference & Exposition*, Anaheim, California, 30 August - 2 September 2010.
2. Boncyk, Wayne C., "Developing a Distributed Power and Grounding Architecture for PnPSat," *IEEE Aerospace Conference*, Big Sky, Montana, 1-8 March 2008.
3. Brewin, Bob, "Air Force working on cheaper plug-and-play satellites," *Government Executive*, 22 January 2008, <<http://www.govexec.com/defense/2008/01/air-force-working-on-cheaper-plug-and-play-satellites/26137/>> (Accessed 21 August 2012).
4. Brinton, T., "USAF Seeks Plug-and-play Satellite Technology," *Defense News*, **24**, 7, 16 February 2009, p. 22.
5. Bruhn, F., Lindegren, R., Lyke, J., Henderson, B.K., Rosengren-Calixte, J., Nordenberg, R., "International Harmonization of Plug-and-Play Technology for Modular and Reconfigurable Rapid Response Nanosatellites," *European Space Agency Small Satellite Systems and Services (4S) Symposium*, Funchal, Madeira, Portugal, 31 May - 4 June 2010.
6. Bruhn, F., Selin, P., Kalnins, I., Lyke, J., Rosengren-Calixte, J., Nordenberg, R., "QuadSat/PnP: A Space-Plug-and-play Architecture (SPA) Compliant Nanosatellite," Paper No. AIAA-2011-1575, *AIAA Infotech@Aerospace*, St. Louis, Missouri, 29 - 31 March 2011.
7. Buffington, R., Kief, C., Erwin, R., Androlewicz, J., Lyke, J., "GENSO, SPA, SDR, and GNU Radio: The Pathway Ahead for Space Dial Tone," Paper No. AIAA-2011-1596, *AIAA Infotech@Aerospace*, 29 - 31 March 2011, St. Louis, Missouri.
8. Cannon, S., "Responsive Space Plug & Play with the Satellite Data Model," Paper No. AIAA-2007-2924, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
9. Center, K., Fronterhouse, D., Martin, M., "The Software Strategy for SPA Plug and Play Spacecraft," *IEEE Aerospace Conference*, Big Sky, Montana, March 2010.
10. Charett, Robert, "Open-Source Warfare," *IEEE Spectrum*, November 2007, <<http://spectrum.ieee.org/telecom/security/opensource-warfare>> (Accessed 21 August 2012).
11. Christensen, J., Cannon, S., B. Hansen, "Automatic Software Generation of ASIM Program Code from an xTEDS," Paper No. AIAA-2010-3549, *AIAA Infotech@Aerospace*, Atlanta, Georgia, 20-22 April 2010.

12. Cook, B., Walker, C., "SpaceWire Plug-and-Play: An Early Implementation and Lessons Learned," Paper No. AIAA-2007-2930, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
13. Dossey, S., Lynaugh, K., Davis, M., Middlestead, R., Lyke, J., Crane, J., Roman, C., "Space Plug and Play Avionics Application Programmers Interface for Radio Devices," Paper No. AIAA-2011-1597, *AIAA Infotech@Aerospace*, St. Louis, Missouri, 29 - 31 March 2011.
14. Duffy, M., Chung, S-J., Bergman, L., "An Evolutionary Architecture for the Automated Conceptual Design of Aerospace Systems," Paper No. AIAA-2011-1632, *AIAA Infotech@Aerospace*, St. Louis, Missouri, 29 - 31 March 2011.
15. Foust, J., "Smallsats and standardization," *The Space Review*, 19 September 2005, <<http://www.thespacereview.com/article/455/1>> (Accessed 22 August 2012).
16. Fowell, S., Taylor, C., "Proposed SOIS Plug-and-Play Architecture and Resulting Requirements on SpaceWire Mapping," *International SpaceWire Conference 2007*, Dundee, Scotland, UK, 17-19 September 2007.
17. Francis, N., Collier, P., and Lyke, J., "Optical Networking for Aerospace Systems Provisioned Through Plug and Play Avionics," Paper No. AIAA-3476, *AIAA Infotech@Aerospace Conference*, Atlanta, Georgia, 20-22 April 2010.
18. Fronterhouse, D., Lyke, J., Achramowicz, S., "Plug-and-play Satellite," Paper No. AIAA-2007-2914, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
19. Fronterhouse, D., Lyke, J. "PnPSat," *International SpaceWire Conference 2007*, Dundee, Scotland, UK, 17-19 September 2007.
20. Fronterhouse D., Center K., Preble J. "Building SPA PnP Satellites," *Proceedings of the 7th Responsive Space Conference*, Los Angeles, California, 27-29 April 2009.
21. Fronterhouse, D.C., Center, K., Strunce, B., Mann, T., Dipalma, J., "Pnpsat-2 SPA Technology Testbed Initial Results and Development Status," *IEEE Aerospace Conference*, Big Sky, Montana, March 2010.
22. Graven, P., Kolcio, K., Plam, Y., Hansen, L.J., "Implementation of a Plug-and-Play attitude determination and control system on PnPSat," *IEEE Aerospace Conference*, Big Sky, Montana, 7-14 March 2009.
23. Hansen, L.J., Graven, P., "A Guidance, Navigation and Control (GN&C) Implementation of Plug-and-Play for Responsive Spacecraft," Paper No. AIAA-2007-2911, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
24. Hansen, L.J., Graven, P., Fogle, D., Lyke, J., "The Feasibility of Applying Plug-and-Play Concepts to Spacecraft Guidance, Navigation, and Control Systems to Meet the Challenges of Future Responsive Missions," *7th International ESA Conference on*

Guidance, Navigation, and Control Systems, Tralee, County Kerry, Ireland, 2-5 June 2008.

25. Hansen, L.J., Graven, P., "A Hierarchy of Guidance, Navigation, and Control Elements for Responsive Space Missions," *IEEE Aerospace Conference*, Big Sky, Montana, March 2010.
26. Holmes, Erik, "Chileworks To Aid in Quicker Execution of Space Assets," *Defense News*, **24**, 8, 23 February 2009, p. 40.
27. Jaffe, P., Clifford, G., Summers, J., "SpaceWire for Operationally Responsive Space as Part of TacSat-4," *International SpaceWire Conference 2007*, Dundee, Scotland, UK, 17-19 September 2007.
28. Jean, Grace, "Can the Air Force Build a Satellite in Six Days," *National Defense Magazine*, July 2007,
<<http://www.nationaldefensemagazine.org/archive/2007/July/Pages/CantheAirForceBuild2583.aspx>> (Accessed 21 August 2012).
29. Kahraman, Mesut "Ozkan , "A Constraint Based Approach For Building Operationally Responsive Satellites," MS Thesis, Air Force Institute of Technology, AFIT/GSS/ENY/08-S02, September 2008.
30. Kief, C., Hansen, B., Mee, J., Christensen, J., "CubeFlow: Training for a New Space Community," Paper No. AIAA-2010-3429, *AIAA Infotech@Aerospace Conference*, Atlanta, Georgia, 20-22 April 2010.
31. Kimmerly, C., McGuirk, P., Rakow, G., Jaffe, P., Klar, R., Bertrand, A., "Application of the SpaceWire Plug-and-Play Protocol," *International SpaceWire Conference 2007*, Dundee, Scotland, UK, 17-19 September 2007.
32. Kleiman, Michael, "Operational satellite assembled in less than two hours," 377th Air Base Wing Public Affairs, 2008.
33. Lanza, D., Lyke, J., Zetocha, P., Fronterhouse, D., Melanson, D., "Responsive Space Through Adaptive Avionics," 2nd *Responsive Space Conference*, Los Angeles, CA, 19-22 April 2004.
34. Lanza, D., Vick, R., Lyke, J., "The Space Plug- and- Play Avionics Common Data Dictionary – Constructing the Language of SPA," Paper No. AIAA-2010-3496, *AIAA Infotech@Aerospace Conference*, Atlanta, Georgia, 20-22 April 2010.
35. Lyke, J., Fronterhouse, D., Cannon, S., Lanza, D., Byers, W., "Space Plug-and-play Avionics," 3rd *Responsive Space Conference*, Long Beach, CA, 25-28 April 2005.
36. Lyke, J., Cannon, S., Fronterhouse, D., Lanza, D., Byers, T., "A Plug-and-play System for Spacecraft Components Based on the USB Standard," *19th Annual Conference on Small Satellites*, Logan, UT, 8-11 August, 2005.

37. Lyke, J., "Space-Plug-and-Play Avionics (SPA): A Three-Year Progress Report," Paper No. AIAA-2007-2928, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
38. Lyke, J., "Plug-and-play as an Enabler for Future Systems," Paper No. AIAA-2010-8660, *AIAA Space Conference*, Anaheim, California, 30 August – 2 September 2010.
39. Lyke, J., Mee, J., Bruhn, F., Chosson, G., Lindegren, R., Lofgren, H., Schulte, J., Cannon, S., Christensen, J., Hansen, B., Vick, R., Vera, A., Calixte-Rosengren, J., "A Plug-and-play Approach Based on the I2C Standard," *24th Annual Conference on Small Satellites*, Logan, Utah, 8-11 August 2010.
40. Marshall, J., Berger, R., Robertson, J., Miller, S., "Reconfigurable and Processing Building Blocks in Responsive Space," Paper No. AIAA-2007-2923, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
41. Martin, Maurice, Fronterhouse, Don, Lyke, James, "The Implementation of a Plug-and-Play Satellite Bus," *22nd Annual Conference on Small Satellites*, Logan, Utah, 11-14 August 2008.
42. McGuirk, P., Rakow, G., Kimmerly, C., Jaffe, P., "SpaceWire Plug-and-Play (PnP)," Paper No. AIAA-2007-2917, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
43. McNutt C., Vick R., Whiting H., Lyke J., "Modular Nanosatellites – (Plug-and-Play) PnP CubeSat", *7th Responsive Space Conference*, Los Angeles, CA, 27-29 April 2009.
44. Mendham, P., Florit, A. Parkes, S., "SPACEWIRE PLUG-AND-PLAY: A ROADMAP," *International SpaceWire Conference 2008*, Nara, Japan, 4-6 November 2008.
45. Moore, G., Holemans, W., Huang, A., Lee, J., McMullen, M., White, J., Twiggs, R., Malphrus, B., Fite, N., Klumpar, D., Mosleh, E., Mashburn, K., Wilt, D., Lyke, J., Davis, S., Bradley, W., Chiasson, T., Heberle, J., Patterson, P., "3D Printing and MEMS Propulsion for the RAMPART 2U CUBESAT," *24th Annual Conference on Small Satellites*, Logan, Utah, 8-11 August 2010.
46. Morphopoulos, T., Hansen, L.J., Pollack, J., Lyke, J., Cannon, S., "Plug-and-Play: An Enabling Capability for Responsive Space Missions," *2nd Responsive Space Conference*, Los Angeles, CA, 19-22 April 2004.
47. Orlando, F., Chalfant, C., "FireRing® (IEEE 1393) a Roadmap to Aerospace Plug and Play," Paper No. AIAA-2007-2920, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
48. Rakow, G., McGuirk, P., Kimmerly, C., Jaffe, P., "Space Wire Plug `n' Play," *IEEE Aerospace Conference*, Big Sky, Montana, 3-10 March 2007.

49. Rojdev, K., Kennedy, K., Williams, R., Yim, H., Hong, T., Studor, G., Delaune, P., Wagner, R., "A Modular Instrumentation System for NASA's Habitat Demonstration Unit," AIAA-2010-8788, *AIAA SPACE 2010 Conference & Exposition*, Anaheim, California, 30 August - 2 September 2010.
50. Romero, J., Czajkowski, D., Lyke, J., Collier, C., Avery, K., "Design of an Optical Appliqué Sensor Interface Module (O-ASIM)," Paper No. AIAA-2011-1474, *AIAA Infotech@Aerospace*, St. Louis, Missouri, 29 - 31 March 2011.
51. Santangelo, A., "OpenSAT(TM), a framework for satellite design automation for responsive space," Paper No. AIAA-2007-2910, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
52. Scott, J., Lyke, J., "Applique Sensor Interface Module: An Enabling Technology for Space Plug-and-Play Systems", *21st Annual Conference on Small Satellites*, Logan, Utah, 13-16 August 2007.
53. Slane, F., Hooke, A., "Space Plug and Play Avionics Standards: Progress, Problems and a View of the Future," Paper No. AIAA-2007-2912, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
54. Some, R., Neff, J., Lyke, J., "Lessons Learned in Building a Spacecraft XML Taxonomy and Ontology," Paper No. AIAA-2007-2913, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.
55. Sorenson, T., Prescott, G., Villa, M., Brown, D., Hicks, J., Edwards, A., Lyke, J., George, T., Mobasser, S., Tyson, S., "KUTESAT-2, A Student Nanosatellite Mission for Testing Rapid-Response Small Satellite Technologies in Low Earth Orbit," *3rd Responsive Space Conference*, Long Beach, CA, 25-28 April 2005.
56. Strunce, R., Eckert, F., Eddy, C., "Responsive Space's Spacecraft Design Tool (SDT)," *4th Responsive Space Conference*, Los Angeles, CA, 24-27 April 2006.
57. Summers, J., "Plug and play testbed to enable responsive space missions," *IEEE Aerospace Conference*, Big Sky, MT, 5-12 March 2005.
58. Sundberg, K., Cannon, S., Hospodarsky, T., Fronterhouse, D., Lyke, J., "A Satellite Data Model for the AFRL Responsive Space Initiative," *20th Annual Conference on Small Satellites*, Logan UT, 14-17 August 2006.
59. Sutherland, Benjamin, "Plug-and-Play Satellites," *Newsweek*, 15 October 2009, <<http://www.thedailybeast.com/newsweek/2009/10/15/plug-and-play-satellites.html>> (accessed 21 August 2012).
60. Van Bruaene, J., "Standards-Based Plug-and-Play Data Distribution," Paper No. AIAA-2007-2901, *AIAA Infotech@Aerospace Conference*, Rohnert Park, CA, 7-9 May 2007.

61. Vera, A., Sibley, M., Ardalan, S., Avery, K., Lyke, J., "Appliqué Sensor Interface Module Based on 90nm Rad- Hard Structured Application- Specific Integrated Circuit," Paper No. 2010-3475, *AIAA Infotech@Aerospace Conference*, Atlanta, Georgia, 20-22 April 2010.
62. Vick, R., Lyke, J., "Development of a Low Power Space Plug- and- Play Avionics Protocol for Simple Devices," Paper No. AIAA-2010-3477, *AIAA Infotech@Aerospace Conference*, Atlanta, Georgia, 20-22 April 2010.
63. Walker, W., Manning, W., McFarland, C., Lyke, J., "Performance Characterization of a Space Plug-and-Play Avionics Appliqué Sensor Interface Module," Paper No. AIAA-2011-1502, *AIAA Infotech@Aerospace*, St. Louis, Missouri, 29 - 31 March 2011.
64. Wegner, P., Kiziah, R., "Pulling the Pieces Together at AFRL," *4th Responsive Space Conference*, Los Angeles, CA, 24-27 April 2006.
65. Williams, A., Palo, S., "Issues and Implications of the Thermal Control Systems on the "Six Day Spacecraft"," *4th Responsive Space Conference*, Los Angeles, CA, 24-27 April 2006.

List of Abbreviations, Acronym, and Symbols

Acronym/ Abbreviation	Description
SDM	System Data Model (formerly Satellite Data Model)
SPA	Space Plug-and-play Architecture

DISTRIBUTION LIST

DTIC/OCP

8725 John J. Kingman Rd, Suite 0944

Ft Belvoir, VA 22060-6218 1 cy

AFRL/RVIL

Kirtland AFB, NM 87117-5776 2 cys

Official Record Copy

AFRL/RVSE/James C. Lyke 1 cy

(This page intentionally left blank)