

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A245 786



DTIC
ELECTE
FEB 12 1992
S D D

THESIS

THE DEVELOPMENT OF A DESIGN DATABASE
FOR THE
COMPUTER AIDED PROTOTYPING SYSTEM

by

Andrew Patrick Dwyer
and
Garry Wayne Lewis

September, 1991

Thesis Advisor:

Dr. Luqi

Approved for public release; distribution unlimited

92-1 109

92-03486

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		4. PERFORMING ORGANIZATION REPORT NUMBER(S)	
5. MONITORING ORGANIZATION REPORT NUMBER(S)		6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School	
6b. OFFICE SYMBOL (if applicable) CS		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION National Science Foundation/NSWC		8b. OFFICE SYMBOL (if applicable)	
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER CCR-9058453		10. SOURCE OF FUNDING NUMBERS	
8c. ADDRESS (City, State, and ZIP Code) Washington, DC 20550		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) THE DEVELOPMENT OF A DESIGN DATABASE FOR THE COMPUTER AIDED PROTOTYPING SYSTEM (U)			
12. PERSONAL AUTHOR(S) Dwyer, Andrew P.; Lewis, Garry W.			
13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM 09/90 TO 09/91	14. DATE OF REPORT (Year, Month, Day) September, 1991	15. PAGE COUNT 377
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	
		Design Database, Engineering Database, Object Oriented Database, Computer Aided Prototyping System,	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The Computer Aided Prototyping System (CAPS) was created to rapidly prototype real-time systems to determine early in the development cycle whether system requirements can be met. The CAPS consists of several software tools that automatically generate an executable Ada model of the proposed system. This thesis describes the development of a design database (DDB) for the CAPS. The DDB is an engineering database that contains all information related to a prototype software design. The DDB enhances the CAPS environment and the prototyping paradigm by providing to the designer the functions of storage, retrieval, viewing, and versioning of prototype components. Garry Lewis is the primary author of chapters I and II and Drew Dwyer the primary author of chapters III and IV. In the joint implementation, Lewis focused on the design database schema and C++/ONTOS issues. Dwyer was responsible for building the command line interface, the hierarchical k-ary data structures and the C++ classes/methods for traversing this structure. Items not covered in the above description were mutually developed.			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Luqi		22b. TELEPHONE (Include Area Code) (408) 646-2735	22c. OFFICE SYMBOL CS/Lq

Approved for public release; distribution is unlimited

**THE DEVELOPMENT OF A
DESIGN DATABASE
FOR THE COMPUTER AIDED PROTOTYPING SYSTEM**

by

Andrew Patrick Dwyer
Captain, United States Marine Corps
B.S., Juniata College, 1980
M.B.A. Webster University, 1983

and

Garry Wayne Lewis
Major, United States Marine Corps
B.A. University of Virginia, 1974
M.B.A. Golden Gate University, 1985

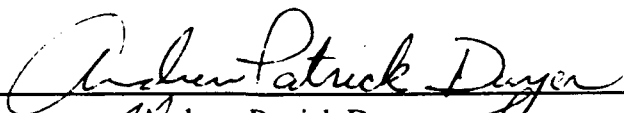
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

Authors:

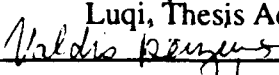


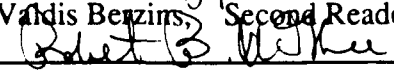
Andrew Patrick Dwyer


Garry Wayne Lewis

Approved By:



Luqi, Thesis Advisor


Valdis Berzins, Second Reader


Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

The Computer Aided Prototyping System (CAPS) was created to rapidly prototype real-time systems to determine early in the development cycle whether system requirements can be met. The CAPS consists of several software tools that automatically generate an executable Ada model of the proposed system. This thesis describes the development of a design database (DDB) for the CAPS. The DDB is an engineering database that contains all information related to a prototype software design. The DDB enhances the CAPS environment and the prototyping paradigm by providing to the designer the functions of storage, retrieval, viewing, and versioning of prototype components.

Garry Lewis is the primary author of chapters I and II and Drew Dwyer the primary author of chapters III and IV. In the joint implementation, Lewis focused on the design database schema and C++/ONTOS issues. Dwyer was responsible for building the command line interface, the hierarchical k-ary data structures and the C++ classes/methods for traversing this structure. Items not covered in the above description were mutually developed.

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail. and/or Special
A-1	



THESIS DISCLAIMER

Trademarks

Ada is a registered trademark of the United States Government, Ada Joint Program Office.

Glockenspiel C++ is a trademark of Glockenspiel Ltd.

ONTOS is a trademark of Ontologic, Inc.

UNIX is a registered trademark of AT&T Bell Laboratories.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. SOFTWARE ENGINEERING	1
B. THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)	2
C. PROBLEM STATEMENT	5
D. OBJECTIVES.	6
E. ORGANIZATION.	6
II. BACKGROUND	7
A. ENGINEERING DATABASE	7
B. OBJECT-ORIENTED DATABASE MANGEMENT SYSTEMS	9
1. Modelling Object Relationships.	9
2. Data Abstraction and Data Hiding.	10
3. Property/Operation Inheritance.	10
4. Method and Trigger Combination.	10
5. Generic Programming.	11
C. ONTOS OBJECT DATABASE.	11
III. REQUIREMENTS SPECIFICATION	14
A. GOALS	14
B. DDB INTERFACES	14
1. Tool Interface.	14
2. End User Interface.	16
C. ENVIRONMENTAL MODEL	20
1. Working Environment	20
2. Version Structure	21
D. BEHAVIORAL MODEL	23
1. Design Constraints.	24

2. Requirements	25
IV. ARCHITECTURAL DESIGN	28
A. GOALS.	28
B. DESIGN DATABASE.	29
1. Informal Solution	29
2. Entities and Their Relationships.	29
3. Pre-defined ONTOS Classes	29
a. Object	30
b. List	31
c. Dictionary.	31
4. Design Database Classes	31
C. TESTING AND EVALUATION	47
D. MAINTENANCE NOTES.	47
V. CONCLUSIONS AND RECOMMENDATIONS	48
A. SUMMARY	48
B. RECOMMENDATIONS FOR FUTURE WORK.	48
1. Variations	48
2. Security	49
3. Evolution Management.	49
APPENDIX A ENVIRONMENTAL MODEL	50
A. STATEMENT OF PURPOSE.	50
B. EVENT LIST	51
C. CONTEXT DIAGRAM	52
APPENDIX B COMMAND INTERFACE.	53
A. PROTOTYPE COMMANDS	53
B. CONFIGURATION COMMANDS	54
C. VERSIONED COMPONENT COMMANDS.	56

APPENDIX C TESTING AND EVALUATION	59
A. NOTES ON TESTING AND EVALUATION	59
B. TEST SCRIPT	59
C. TEST RESULTS	71
APPENDIX D CODE	126
A. PREPARATION NOTES	126
B. PRINTING NOTES	126
C. MAINTENANCE	126
APPENDIX E SETUP FILES	348
A. ONTOS HEADERS/MAKEFILES PREPARATION NOTES . . .	348
B. PRINTING NOTES	348
C. MAINTENANCE	348
LIST OF REFERENCES	356
INITIAL DISTRIBUTION LIST	359

LIST OF FIGURES

Figure 1. Process Model for Software/System	3
Figure 2. Computer Aided Prototyping Environment	15
Figure 3. DDB User Interface (Search for Component)	17
Figure 4. DDB User Interface (View PSDL Source)	18
Figure 5. DDB User Interface (Edit Prototype))	19
Figure 6. DDB User Interface (Edit Prototype Panel)	20
Figure 7. Threaded Components.	21
Figure 8. CAPS Files in Versioned Component	22
Figure 9. Basic DDB/CAPS Functions.	24
Figure 10. Tool/User Interface.	25
Figure 11. Design Database Data Flow Diagram	26
Figure 12. Design Database Entity Relationship Diagram -High Level	30
Figure 13. Design Database Entity Relationship Diagram - Low Level	31
Figure 14. Design Database Descendants of Class Object	33

ACKNOWLEDGMENTS

Sincere thanks to the following contributors of this thesis effort:

To our families:

Words can not express the support, devotion, and patience we received from them. Many thanks for your prayers and encouragement;

Professors Luqi and Valdis Berzins:

For their excellent guidance and support;

Professor Lawrence Williamson:

For his many tireless hours, depth of knowledge, and devotion to the cause;

Captain Patrick Barnes, USAF:

For his overall knowledge of the CAPS system and his development of a user interface for the design database.

Albert Wong and Rosalie Johnson:

For outstanding technical support and encouragement;

Süleyman Bayramoglu:

For his extensive knowledge of Unix and various application programs, and his willingness to impart that knowledge to others.

I. INTRODUCTION

A. SOFTWARE ENGINEERING

Each year billions of dollars are allocated for the development and maintenance of progressively more complex weapons and communications systems [Ref. 1:p. 14]. These systems increasingly rely on information processing using embedded computers systems. Satellite control systems, missile guidance systems and communication networks are examples of embedded systems. Correctness and reliability of these software systems is critical to the reliable operation of national defense systems. Moreover, software development of these systems is an immense task with increasingly higher costs and potential for misdevelopment.

Studies conducted in the early 1970's showed that computer software alone comprised approximately 46 percent of the estimated total DoD computer cost. Of this cost, 56 percent was devoted specifically to embedded systems [Ref. 1:p. 14]. In spite of the tremendous cost, there were examples of software systems produced that exceeded estimated development cost, fell behind the production schedule and were delivered not fully functional.

Software engineering developed in response to the need to design, implement, test, install and maintain more efficiently and correctly larger and more complex software systems. Numerous methodologies have been introduced to support software engineering. The two major approaches which underlie these different methodologies are the phased refinement and prototyping method of development.

The predominant model for current application development is the phased refinement approach. In this approach, all system functionality is specified in the first step of development, and subsequent implementation phases add prescribed design details. This

approach is criticized for its high cost of maintenance, failing to abstract tasks early in the development process, and for complications in system integration. Prototyping, as an evolutionary system development paradigm in which a number of nonstandard concepts work together, promises to achieve effective evolution of integrated hardware/software systems.

Prototyping is the process of developing a scaled-down version of a system to use in building a full-scale system. Computer-aided rapid prototyping promises to provide a means for building a scaled-down version of a system more quickly than using conventional approaches. The final product of the prototyping activity is a working model that can be used for many purposes, such as requirement validation, feasibility study for a complex system, and functional specification of a system design. [Ref. 2:p.9] Figure 1 illustrates the role of prototyping in software development..

The process model differs from the traditional phased approach in that it concentrates on the hard problem of system development, namely; requirement specification, and design rather than coding. Equally important, validation, evaluation, and hardware/software trade-off analysis are all part of the development process--they do not just follow completion of the development in each phase. Rapid prototyping works by providing continuous feedback information as rapidly and efficiently as possible. [Ref. 2:p. 9]

B. THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

The evolution development of a software system design requires an integrated design support environment. A typical structure for a design environment that supports the rapid prototyping paradigm and the evolution process model is that provided by the CAPS[Ref. 4:p. 15].

CAPS is a rapid prototyping environment which includes the ability to prototype hard real-time systems. This approach to rapid prototyping uses a specification language called

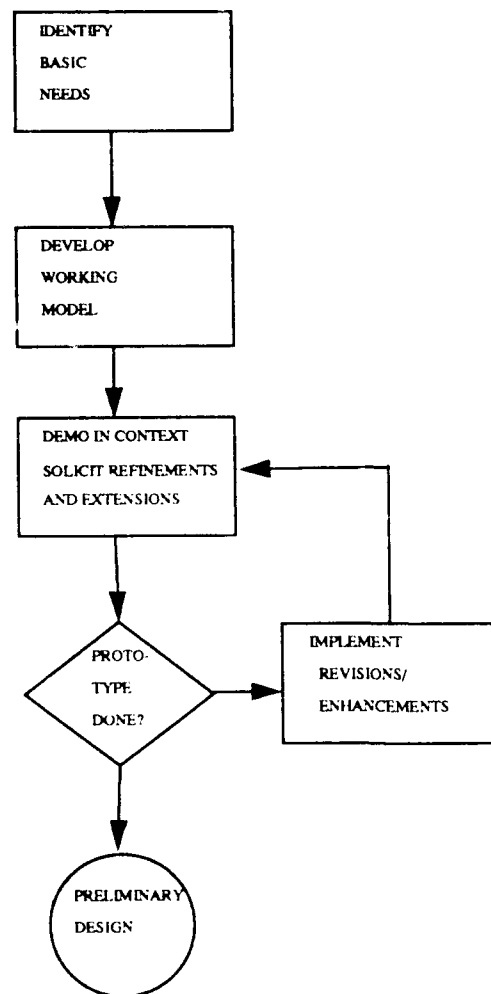


Figure 1. Process Model for Software/System [Ref. 3: p.7]

Prototyping System Design Language (PSDL) and an integrated set of prototyping tools. The tools are integrated through the user interface. The primary tools in CAPS may be divided into three main systems [Ref. 4:p. 15]. The subsystem and their tools are:

The User Interface which is composed of the following tools:

- Graphic Editor
- Syntax Directed Editor
- Browser
- Expert System

The Software Base System which is composed of the following tools:

- Software Design Management System
- Design Database
- Software Base

The Execution Support System which is composed of the following tools:

- Translator
- Static Scheduler
- Dynamic Scheduler
- Debugger

The Graphic Editor is a tool which permits a designer to specify a portion of a PSDL prototype using graphical objects to represent the system. Graphical objects include operators, inputs, outputs, data flows, and self loops on operators. All graphic objects are named and may have time constraints associated with them.

The Syntax Directed Editor is used by the designer to enter the text portions of the prototype design not represented by the graphic editor and to ensure that the prototype is syntactically correct PSDL.

The Browser provides a means for the designer to view reusable components in the software base.

The Expert System provides a paraphrasing capability that generates English text descriptions of PSDL specification. This tool permits users who are unfamiliar with the PSDL language to evaluate a prototype.

The Software Design Management System manages and retrieves the versions, refinements, and alternatives of the prototype in the Design Database and the reusable components in the software base.

The Design Database contains PSDL prototype descriptions for all software projects developed using CAPS.

The Software Base contains PSDL descriptions and implementations for all reusable software components developed using CAPS. [Ref. 5]

The Translator generates high level code from the PSDL prototype which binds the reusable components from the software base to the executable prototype. [Ref. 6]

The Static Scheduler attempts to allocate time slots for the representation of PSDL operators with real-time constraints before the prototype is executed. If the allocation succeeds, all operations are guaranteed to meet their deadlines. [Ref. 7]

The Dynamic Scheduler invokes representation of operators with real-time constraints at run-time to occupy time slots which are not used by operators with real-time constraints. The time slot which the dynamic scheduler uses are "slack times". Dynamic scheduling occurs during execution of the prototype. [Ref. 7]

The Debugger allows the designer to interact with the execution support system. The debugger has facilities for initiating the execution of a prototype, displaying execution results or tracing information of the execution and gathering statistics about a prototype's behavior and performance.

Prior to the work described in this thesis, implementations had been developed for the graphic editor, translator, static scheduler, and user interface. The design of a debugger has been defined. Feasibility studies had been conducted for the syntax directed editor, design database and the software base.

C. PROBLEM STATEMENT

Vast amounts of evolving data are created in the design of hard real-time systems. The data must be managed so that it can be stored and retrieved according to the needs of a team of design engineers. In CAPS, the Design Database (DDB) must manage the storage and retrieval of the PSDL program. The DDB must be a specialized DBMS which will store PSDL specifications in a hierarchical format and manage all support files associated with a software design project. [Ref. 8:p. 5]

The DDB also provides concurrency control functions that allow multiple designers to update parts of the prototype without unintentional interference. In the interest of minimizing delay, the design database will not lock out read-only access to any part of the design, even while the design is being updated. Instead, the system will allow examination of the previous version of the component, with a warning that a new version is currently in preparation. On request, the system will provide information about the reason for modification of the component (such as a new or modified requirement).

D. OBJECTIVES

The objective of this thesis is to develop an object-oriented design database for the CAPS. The design database must be accompanied by a user interface that enables the designer to perform routine database tasks such as opening, closing, querying, storing, and retrieving operations.

E. ORGANIZATION

Chapter II contains a survey of the requirements for an engineering database with an emphasis on the object-oriented approach. An introduction to ONTOS, a state of the art object database will conclude Chapter II. Chapter III contains the design of the database objects. Chapter IV shows the design database implementation details using C++ and ONTOS. Conclusions and recommendations will be presented in Chapter V.

II. BACKGROUND

A. ENGINEERING DATABASE

An engineering database should provide the following facilities to support computer-aided software development environments:

- Persistence
- Concurrency control
- Version control
- Reuse of past design objects
- Configuration control
- A wide variety of data storage
- Guarantees data will not be corrupted due to system or media failure

A fundamental feature provided by a database is persistent storage. If objects are persistent, this means the objects in the database will exist after the process that created them has terminated. [Ref. 9:p.472]

In a design environment, one usually has to keep data on several design alternatives, revisions, design stages, and so on. Hence information about the same semantic entity has to be recorded more than once. If versioning is implemented in the DBMS's storage subsystem, then it can be done on fine granularity components. When creating a new version, only those components that have changed since the previous version need to be stored. In the new version, unchanged components can be represented by pointers to their previous versions. This is efficient in storage because only the changed components are repeated in the new versions. It is also efficient in execution time; since versions are not encoded they can be retrieved directly instead of being reconstructed using change logs, as in RCS and SCCS. [Ref. 10:p. 168]

The goals of configuration management include recording the development history of evolving systems, and aiding the management of the systems in guiding and controlling

their evolution [Ref. 11:p. 918]. A structure that captures the evolution of a design in terms of its hierarchical decomposition, alternative implementation and revision history, is crucial to the effective reuse of past design objects.

The partitioning of the design task across levels of abstraction and between many design engineers requires the design database to support concurrent access and implement concurrency controls. The issue of concurrent access for a design is significantly different than for a conventional database application due to the different lengths of transactions that each must support. In a design environment, individual design objects may be "checked-out" for long periods of time [Ref. 12:p. 66]. However, over the course of some transactions other designers may require "access" to a design object that is already checked-out. The concurrency control system must therefore support an object locking mechanism that remains in the database after the current process has ended and permits a designer to view objects already checked out to be operated on by another designer.

The design database must store a variety of *formatted, unformatted, and variable-length* objects such as program text. In addition to program text, the design database must store management information and natural language descriptions for documentation/historical purposes. One of the criticisms of relational DBMS is that they are inadequate for the data handling requirements of a support environment.

Securing data stored in a design database is another concern [Ref. 13: p. 37]. Some type of mechanism must exist whereby access to objects in the database can be restricted. Protecting high-value data (a civilian company's proprietary information, or sensitive military data) from compromise requires a system that is sophisticated enough to distinguish different user access privileges and appropriately grant or deny access to a particular object in the database.

Finally the database must ensure that data will not be corrupted due to system or media failures.

B. OBJECT-ORIENTED DATABASE MANGEMENT SYSTEMS

Object-oriented database management systems (OODBMSs) in particular offer several features that promote rapid prototyping. One of the major advantages of the object-oriented paradigm is increased modeling power. Objects and their relationships in OODBMSs can be aligned very closely to objects and their relationships in the real-world. On the other hand, conventional data modeling paradigms such as file formats or relational models require considerable effort to force real-world objects into fixed programming constructs. Thus, a semantic gap exist between the way information is stored and the way it is used. Relational databases, in particular, are inadequate for storing complex information structures such as those in the CAPS. [Ref. 14:p. 32]

OODBMSs typically provide a set of predefined system types, such as set, queue, stack, list, and ordered dictionary. This simplifies the modeling effort as the designer can concentrate on the problem at hand and not clutter the solution with "data structure" specific procedures.

1. Modelling Object Relationships.

Using a limited set of constructs to model diverse concepts inevitably leads to loss of exact meaning. OODBMSs provide several mechanisms to model different relationships among objects. Object specialization/generalization refers to the ability to organize objects in an is-a-kind-of hierarchy. Classification is the ability to relate an object to a group of objects via the is-an-instance-of relationship. Aggregation allows the programmer to model an object as an aggregate of its constituent objects. This type of relationship is known as the is-a-part-of relationship. [Ref. 14:p. 32]

2. Data Abstraction and Data Hiding.

The advent of third-generation programming languages points out the need for a clear distinction between the storage structures associated with the data and the logical structure of the information. Data abstraction refers to the extreme case of this separation, where access to the storage structure may only be gained through a set of predefined operations. Data hiding insulates the programmer from the actual structure where data is stored. In OODBMSs, completely encapsulating an object with its data and operations forces the programmer to use the correct operations on all objects. In addition, a programmer need only know about the objects that his or her code uses. [Ref. 14:p. 33]

3. Property/Operation Inheritance.

Property/operation inheritance implies that an object type automatically has all the properties/operations of its parent type. This is one of the fundamental differences between OODBMSs and relational databases. [Ref. 14:p. 33]

From the standpoint of rapid prototyping, operation inheritance is perhaps more valuable than property inheritance. A programmer does not need to reimplement the behavior that a type shares with its supertype. More importantly, this allows a programmer to assign default behavior to objects. In fact, once the subtype-supertype link is established in the is-a-kind-of hierarchy, a large amount of code automatically become available to the object, even before a single line of code specific to the new object type has been written. [Ref. 14:p. 33]

4. Method and Trigger Combination.

Method and trigger combination invokes writing several small modules of code and using them in different combinations to form methods for different types and situations. In conventional programming, this would involve many checks which would make the code inefficient. However, with OODBMSs, the automatic dispatching mechanisms (see the next section) allows programmers to combine modules by triggering them at the

appropriate places. Each object-specific operation module is designed to handle its additional set of properties and then invoke the corresponding module of its supertype.

5. Generic Programming.

In generic programming, programmers write code modules as general as possible so they can be used by different types of objects. Two essential features of the object paradigm--polymorphism and access to metainformation--help generic programming. Polymorphism is the ability to automatically dispatch a call to an appropriate routine according to the type of parameters passed. This feature makes code upward compatible and resilient to modifications [Ref. 14:p. 34]. Another characteristic of OODBMSs that makes a program resilient to the addition of new types is the accessibility of metainformation. The user-defined types are actually objects compiled into the database. They are therefore available to a program just like any other project. The advanced exception-handling capability of OODBMSs comes in handy in rapid prototyping. Typically, a large amount of code in an application's final version deals with erroneous input data or other anomalous situations. Programmers would like to avoid such detailed error handling in early prototypes. If exceptions are also treated as objects, a programmer can initially write a simple exception handler for the most general type of exception. The programmer can then gradually refine this default exception handler as the prototype grows.

C. ONTOS OBJECT DATABASE

We used the ONTOS Object Database as the database engine for the design database. ONTOS, and its predecessor Vbase, are products of Ontologic, Inc. Vbase was used at the Naval Postgraduate School to test the feasibility of a design database after completion of a conceptual level design [Ref. 8:p. 9].

We conclude this section with a description of the functionality provided by the ONTOS Object Database.

ONTOS supports the full object model. The database schema represents object classes, data members (properties) and member functions (operations). Any class definition which can be specified in C++ can be stored in ONTOS as an ONTOS Type.

ONTOS supports both the single and multiple inheritance model of object oriented programming. The class model that is used in the C++ application is represented in the database as object schema. ONTOS schema information is stored directly in the database as object data.

ONTOS provides three interfaces for use by the application programmer: a C++ interface, an Object SQL interface, and a programmatic schema manipulation facility. We are primarily concerned with the C++ interface. ONTOS provides an interface to C++ which is simple to use. It provides a transparent database interface which is generated by ONTOS utilities.

ONTOS uses a standard client-server architecture. C++ client applications interact with a logical ONTOS database server. The ONTOS database server provides object storage, transactions, concurrency control and other database services. C++ client application make requests to the database server to access and store objects. When a client application requests an object, the system automatically translates the object into its C++ in-memory representation and places the object in the C++ client application's in-memory process heap.

ONTOS provides Aggregate classes including Set, List, Array, and Dictionary. Sets adds the functionality of mathematical sets to C++. Sets are unordered, unkeyed Aggregates with no duplicate members. Lists provide the abstraction of linked list; logically, members are stored serially, in a chain. Each position in the chain is numbered, starting at position zero. An Array is an Association of declared size whose keys are a continuous range of integers and whose members are Entities. Dictionaries are

Associations whose keys and members are Entities and whose size is unrestricted. Dictionaries may be ordered or unordered.

ONTOS supports the standard, locked-based transaction model which is common to all major database systems. The transaction protocol supports transaction start, checkpoint, commit and abort. Checkpoint allows a transaction to commit the changes made up to checkpoint and then to continue with the transaction. ONTOS transactions also support transaction journaling. Journaling maintains a log of changes made to the database within a transaction, which can be used to provide roll-forward recovery after a server process failure.

In addition to the standard transaction model, ONTOS supports high-concurrency and shared transactions for workgroup applications, as well as nested transactions with transaction-based undo capabilities.

Onto supports locking at the level of object, aggregate and page. Object-level locking provides locks on individual objects, thus providing the highest possible concurrency. Aggregate -level locking allows any aggregate of objects to be locked as a group, to provide higher-granularity locking and concurrency control strategies. Page-level locking provides optimum locking performance for large groups of objects physically clustered on disk.

The high level of abstraction provided by C++, together with ONTOS' complete support of the object model, matched against the requirements for a engineering database gave us complete confidence in our choice of tools for implementing a design database for CAPS.

III. REQUIREMENTS SPECIFICATION

Here we define and present a summary of the requirements specification for the DDB. This set of requirements was developed using the Berzins and Luqi model of Software Engineering. This model requires a **problem statement**, development of an **environmental model** and **goals**, and consideration of the **constraints** imposed on the system. [Ref. 9:p. 24]

An additional methodology providing complimentary structure and style is Yourdon [Ref. 15]. In this approach the designer develops an Environmental Model and a Behavioral Model. The environmental model details interactions between the DDB and the user. The users in our model are the tool interface of the CAPS system and the end user (designer). The tool interface responds to the needs and queries of the end user and other tools in the CAPS system. The behavioral model describes the internal mechanisms at work in the DDB system.

A. GOALS

The goals of the DDB are as follows:

- Provide functionality which supports the needs of a prototype designer using the CAPS system.
- Develop a system which encapsulates data and functions together
- Provide flexibility for future development and change

B. DDB INTERFACES

1. Tool Interface

The tool interface is the glue which binds the individual CAPS tools together. A picture containing the tool interface and its relationship to the other tools is provided in Figure 2. The tool interface interprets and reformulates requests sent from one tool to

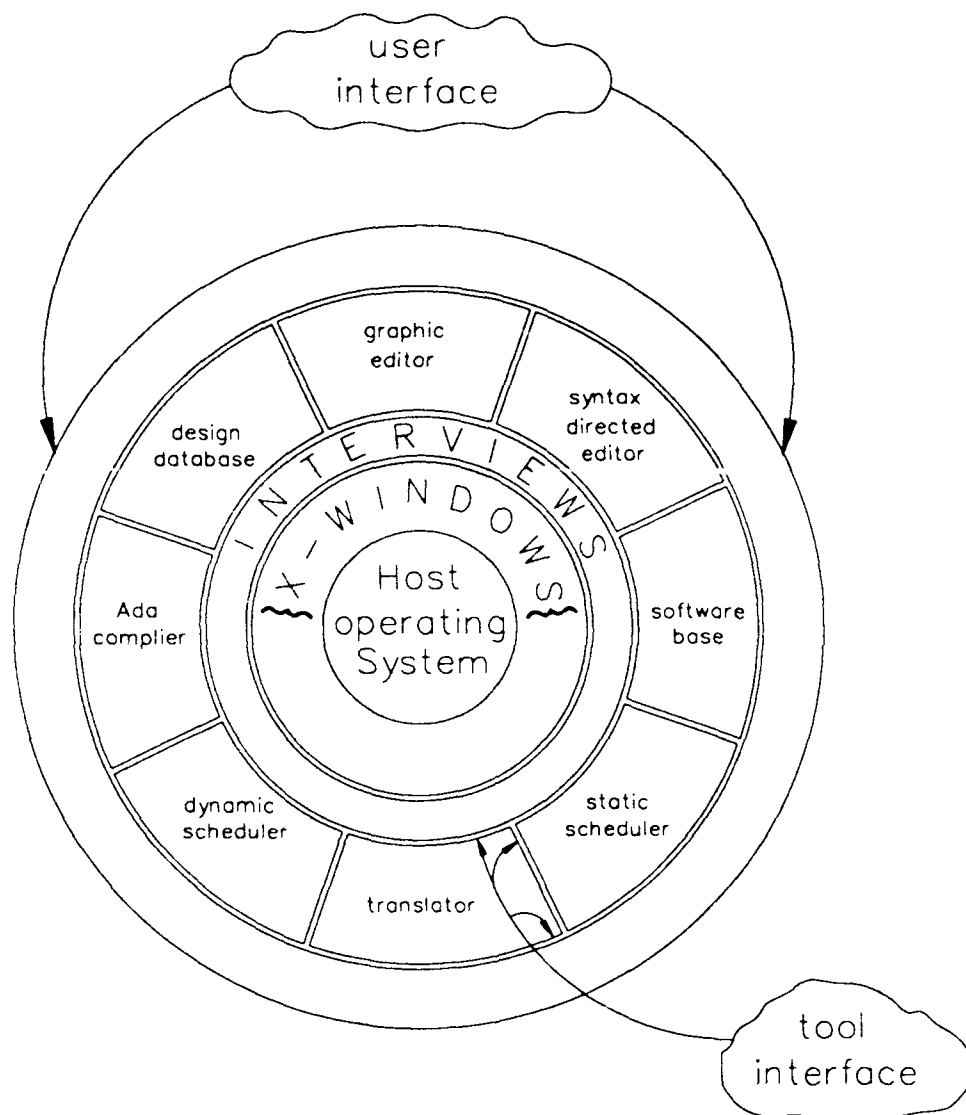


Figure 2. Computer Aided Prototyping Environment [Ref. 3:p. 16]

another - modifying the request to achieve the desired output. A typical sequence of requests is shown in Table 1 in informal language.

TABLE 1: TYPICAL TOOL INTERFACE INTERACTION

Tool	Converses with	Operation
Graphic Editor	Tool Interface	"I want to view operator X"
Tool Interface	Design Database	"Return operator X in Read Mode"
Design Database	Tool Interface	"Operation Complete"
Tool Interface	Graphic Editor	"Resume Operation"

In the previous design of CAPS, if the designer had to communicate with a tool, that tool provided its own end user interface [Ref. 3:p. 21]. However, the DDB responds to commands only from the tool interface. The user interface required to query the DDB and store, retrieve, or manipulate prototypes is important for extending the DDB's services to the end user. A graphical user interface has been developed as a separate tool that interacts with the DDB via the tool interface. This end user interface extends to the user those functions defined in chapter IV of this thesis. The end user interface is discussed briefly in the next section.

2. End User Interface

Due to the scope of the DDB's requirements, we need to separate the DDB's functionality from the user interface functionality. For this reason the end user interface requirements of the DDB were removed and developed separately from the functions provided by the DDB. A graphical user interface developed at the Naval Postgraduate School allows the designer to take full advantage of the functions provided by the DDB [Ref. 14]. The user interface runs under the control of the tool interface. An example of the screens which a user will see when he requests access to prototypes stored in the DDB is illustrated in Figures 3 through 6. These figures illustrate the sequence of panels a designer will see/use when generating the following DDB commands:

- List the names of the initial decomposition of prototype operators stored in the DDB and search for a component contained inside of the composite

component "user_interface"

- View a component's PSDL Source
- Select and Edit a prototype's attributes

File	Edit	View	Select	Quit
Database: ddb53				
Prototype: c3i				
Configuration: sparc				
Operator		Version		
↑ .. commons_interface commons_links navigation_system sensor_interface sensors track_database_manager user_interface weapons_interface weapons_systems ↓				
		1		
		1		
		1		
		1		
		1		
		1		
		1		
		1		
		1		

Figure 3. DDB User Interface (Search for Component)

The user moves the selection bar to the desired versioned component (Operator). After the operator (user_interface) is highlighted, double click on that component to see the decomposition of that operator. The resulting view of the decomposition of the user_interface operator is shown in Figure 4.

After the PSDL operator **message_editor** is highlighted (single clicking on that operator) the designer selects the pull-down menu option File and moves the selection bar

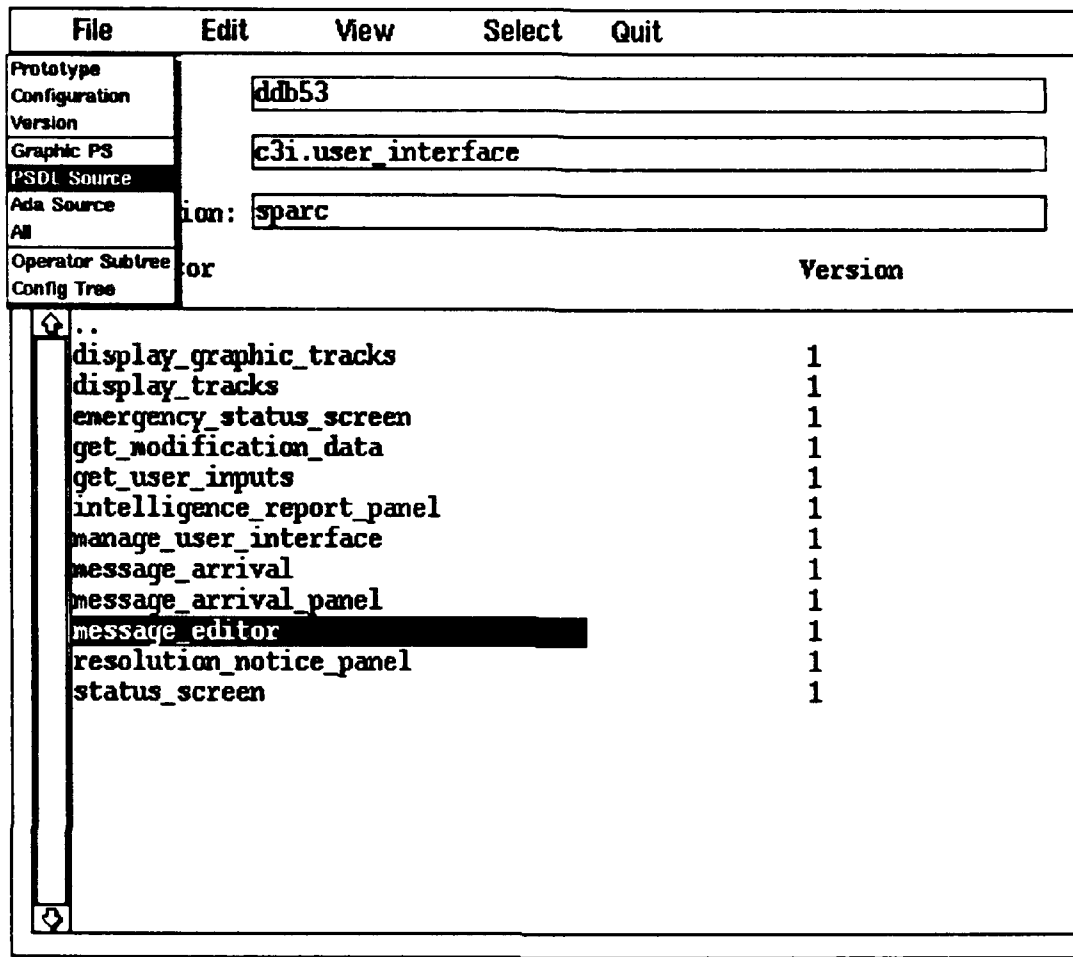


Figure 4. DDB User Interface (View PSDL Source)

to PSDL Source. When that option is activated the user is able to view the contents of the PSDL Source for the message_editor (not shown).

In Figure 5 the user highlights the prototype c3i and selects edit prototype. The results of this action (Figure 6) present the user with an edit screen which allows him to edit various data fields of the prototype. The composition of prototypes and other classes implemented in the DDB will be explained in Chapter IV.

This method of developing the user interface (DDB->tool interface->user interface) enhances modularity and provides flexibility to the overall CAPS system. Consequently,

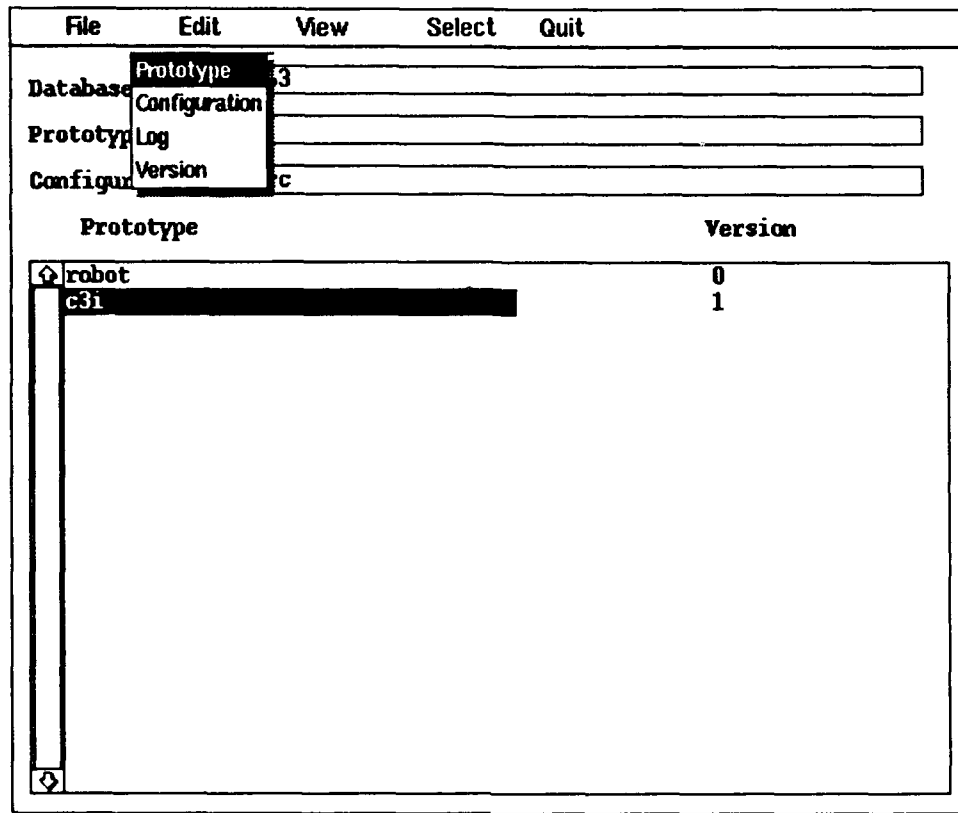


Figure 5. DDB User Interface (Edit Prototype))

whatever form the tool interface and other tools assume in the future, the functions of the DDB shall require little modification. Designing the DDB in this manner permits porting the CAPS system to different graphical user interfaces existing on a wide variety of system platforms.

The above examples represent a small portion of the capabilities of this engineering database. These functions are further defined later in this thesis. Hereafter in this chapter the term "user" will be used generically to refer to the tool interface and/or prototype designer (end user).

Save Quit	
Prototype: c3i	Date: Thu Sep 12 16:00:00 1991
Leader:	Marianne Aiello
Description:	
This project deals with the command and control functions in the Aegis Cruiser line.	
Expected completion date is December, 1992.	
Team Members assigned to develop this Prototype	
John 646-5123 Henry 646-7815 MaryAnne 646-0917	

Figure 6. DDB User Interface (Edit Prototype Panel)

C. ENVIRONMENTAL MODEL

The DDB statement of purpose, event list, and context diagram are contained in Appendix A. A summary of the model is presented in the remainder of this chapter.

1. Working Environment

Prototypes are developed in a system work area for CAPS development specified by a case sensitive Unix environment variable (\$PROTOTYPE) defined by the user . The value of this variable should be the path name of a unique Unix directory. The DDB reads this system variable to ascertain where to output prototype components. Components are

immutable snapshots of prototype development efforts. Components may be composite - in which case they contain other components, or they may be atomic - in which case they do not have a decomposition. Atomic components may be transformed to composite components in future decompositions. Once a prototype is stored into the DDB, the database copy may not be altered. To modify a prototype, a copy of the prototype must be checked out of the DDB and put into the user's local area. The user then modifies the working copy. When this copy is significantly altered and certified as an updated version it is checked back into the DDB as a new version of the prototype. This process is called threading components [Ref. 11].

2. Version Structure

Each component represents a PSDL operator or type. Every component in the DDB has its own unique thread. This concept is illustrated in Figure 7.

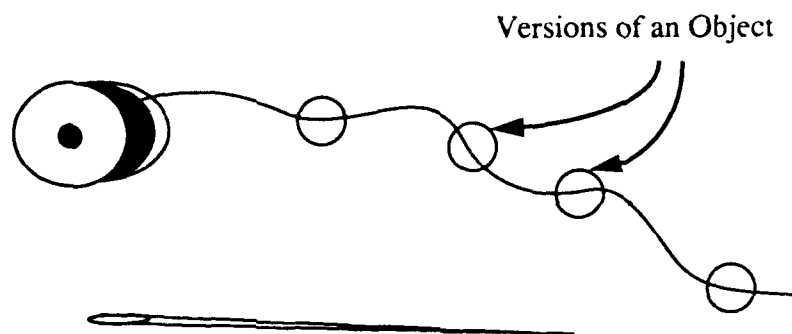


Figure 7. Threaded Components

Versions of a component are ordered by a thread. Each component in the database may reference several types of files generated by various CAPS tools. Figure 8 shows the potential set of files generated for a single component in the DDB.

Inserting a component in a thread and storing it into the database can be modeled as creating a node in a multi-level acyclic graph. In the fully developed model, threads may split at

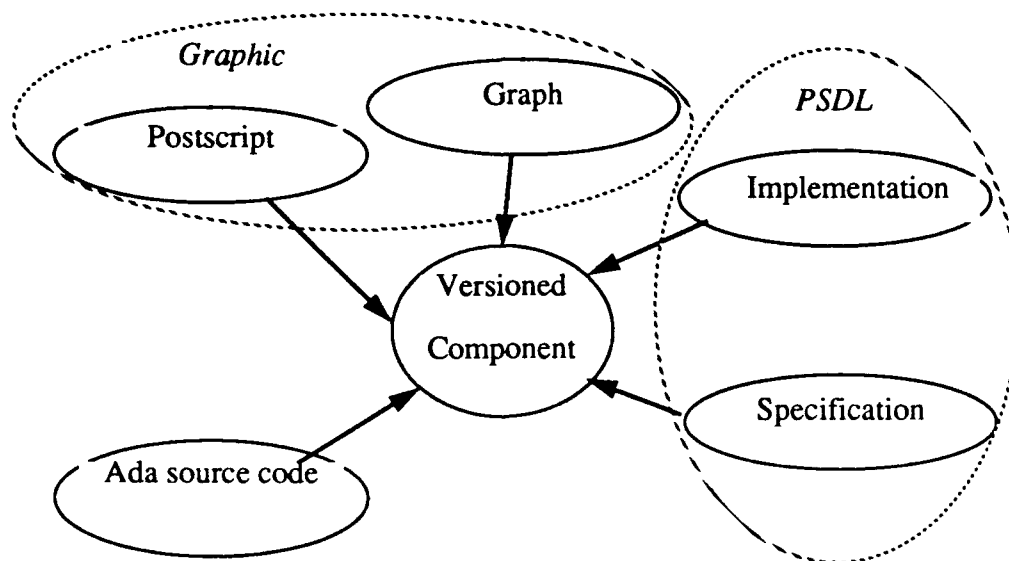


Figure 8. CAPS Files in Versioned Component

any point into variations [Ref 11]. In the current implementation, there is no provision for a segmented join of separate variations, although the groundwork has been laid in the design of object oriented structures which supports a transition to the full model. Decomposing components is an essential requirement for modeling the problem domain. This decomposition produces a multi-level k-ary hierarchy that must be navigated effectively to view and modify any version object upon demand. For this reason "walking the graph" [Ref. 17] is essential.

Three of the primary entities identified for the DDB - a **prototype**, a **configuration**, and a **versioned component** will be discussed in the next section.

Taken as a whole, CAPS, PSDL, and the DDB provides the software developer the ability to store, retrieve, and manipulate an unlimited number of prototypes and components. Working jointly with the other tools in the CAPS system, the DDB provides a capability never before available in computer aided prototyping. This enables the

development of large, real time, or embedded application prototypes more quickly and efficiently.

D. BEHAVIORAL MODEL

Just as a motion picture is a collection of still pictures, a prototype is a collection of versioned components. Our ability to understand a movie is directly related to how well we can grasp the relationship and flow from one frame to the next. Taken together as a movie, these frames can convey something of power and beauty. Developing a software prototype is analogous to understanding a movie. With the right grasp of the tools available in computer aided prototyping, we can form truly productive prototypes. It is the relationship of one version to the next that conveys the direction of future development. As this transformation is captured and tools such as this DDB become fully mature, changes in one component that are dependent on another will be largely realized by automated transformations internal to the tool. These DDB consistency checks ensure that all components interact through their relationship in a database hierarchy. It is for this reason that cataloging and indexing individual CAPS components is essential. This is referred to as Evolution Management. [Ref 11]

The basic prototype functions are illustrated in Figure 9.

The DDB supports the viewing, storing, and retrieving of prototype components in a central design repository. Since prototypes are usually multi-level decomposed subtrees, the DDB provides a mechanism for viewing the complete hierarchy of a prototype components. The DDB stores components required by the other CAPS tools indefinitely. The DDB has no destructive capabilities on the attributes displayed in Figure 8. Once prototypes are posted to the DDB they will never be removed - although a capability to migrate prototypes to archival storage such as tape is planned. The DDB supports the concept of configuration control on a desired slice of the entire prototype history. The DDB does not store repetitive copies of individual components which do not have different versions.

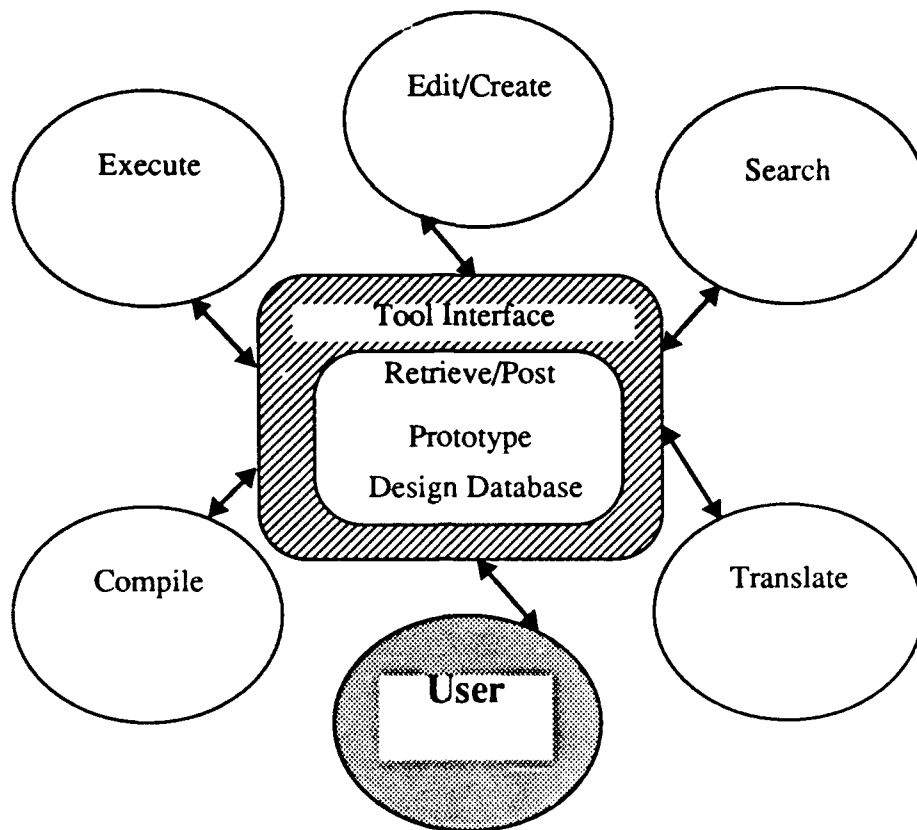


Figure 9. Basic DDB/CAPS Functions

1. Design Constraints

The DDB processes commands from the tool interface. The manner in which data is passed between the tool interface, the DDB, and the end user interface is illustrated in Figure 10. The tool interface controls the user interface. The DDB communicates indirectly with the end user through the user interface's handling of output from the main function calls to the DDB. Since the functionality of the DDB is separated from the user, security is enhanced and there is less chance for error.

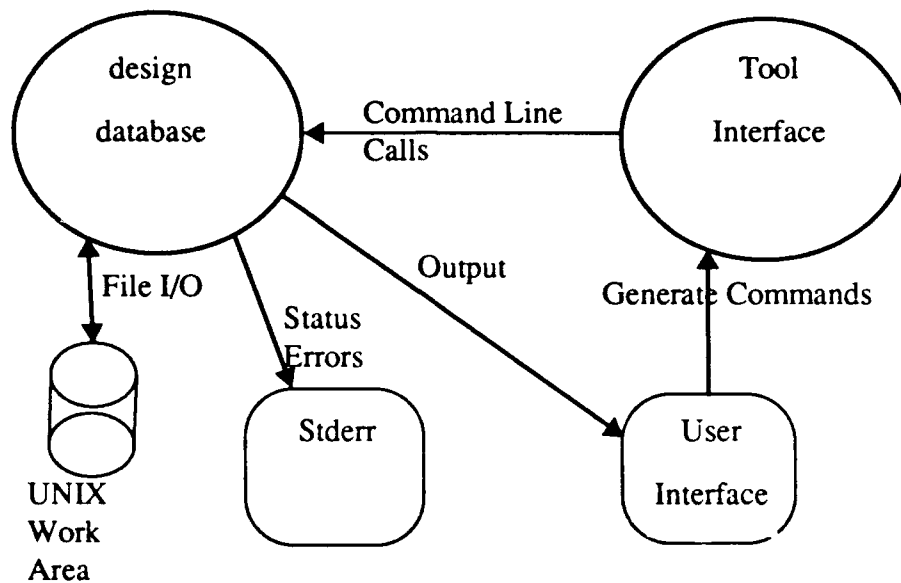


Figure 10. Tool/User Interface

2. Requirements

The user interface captures the standard output and displays it to the user in a window. The data flow diagram of the DDB is shown in Figure 11. The DDB first parses the command line to verify that the command contains a legal function tag and the appropriate number of arguments. A listing of the valid commands to the DDB is contained in Appendix B. After parsing the command, the DDB prepares the ONTOS DBMS for storage and/or retrieval by issuing specific ONTOS DBMS calls. Next the DDB reformulates the command and initiates the sequence of functions required to perform the transaction. At this point, the DDB will perform a query of persistent objects stored in the database or will create and input components designated to be checked into the database. The remaining step displays confirmation and/or displays the data to standard output (stdout).

The end user can then choose to work with one of the other tools in the CAPS system for an indefinite period of time. When the designer/manager is ready to check a revision of the prototype back into the DDB he will provide the tool interface a prototype name and have the tool interface command the DDB to search

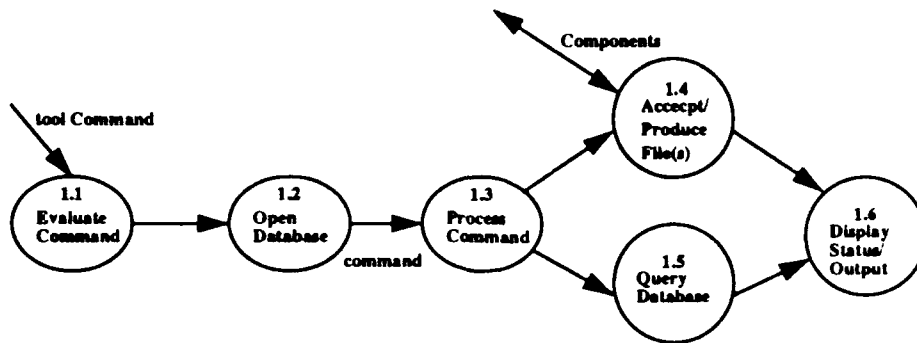


Figure 11. Design Database Data Flow

the working directory defined by \$PROTOTYPE to check modified or newly created components into the database as a new version of the prototype.

The capabilities provided to the user will at a minimum contain the following high level operations:

- Insert new prototypes into the DDB
- View existing prototypes available in the DDB
- View/Update prototype attributes
- Insert new configurations into the DDB
- View existing configurations available in the DDB
- View/Update configuration attributes
- Insert new components into the design DDB
- View versioned components available in the DDB
- Retrieve selected versions of components from the DDB
- Prevent others from versioning components which are currently evolving
- Update previous versions spawning alternate thread histories
- View/Update component descriptions in the DDB

In addition to the above capabilities, the DDB will allow the tool interface to:

- Override component locking mechanisms
- View component lock status

- View a worker which currently has a component locked
- List decomposition of operator/type components and their descendants
- Attach any component to a configuration
- Navigate the hierarchy from any component directly or through a configuration.

More generic database management capabilities are provided by the ONTOS object oriented database. These DBMS capabilities are considered vital to the successful implementation of any database management system. For further information pertaining to these and other ONTOS issues the reader is referred to Chapter II and the ONTOS (version 2.01 beta) users' manual.

IV. ARCHITECTURAL DESIGN

Much of the analysis and research in this area dictates an object oriented approach for classifying the schema and functions of a prototyping storage and retrieval system. To version components which carry information relating to specifications of a software prototype the DDB is modeled in terms of its major classes TEXT_OBJECT, COMPONENT, VERSIONED_OBJECT, THREAD, CONFIGURATION, and PROTOTYPE. In this chapter we define these classes, their behavior and attributes, and the operations they perform.

A. GOALS

The process of developing this application utilizes the object-oriented Programming concepts of specification, implementation, and refinement [Ref. 18:p. 5]. First we collect the requirements in terms of user needs. The result of this phase is documented in Chapter III. The next step is to develop a conceptual solution to the requirements. We then convert these conceptual ideas into concrete classes and operations. This is accomplished by first converting the entity-relationship (ER) diagram to a functional specification.

From the functional specification we determined that a command line interface would best satisfy the requirement for separating the functionality of the tool from other CAPS tools and from the end user. The command line interface divides the functionality into three distinct classes - functions dealing with prototype level operations, those concerned with configurations, and those dealing with versioned component objects. The complete command line interface is contained in Appendix B.

Now we develop an informal solution followed by a formal solution to the stated design requirements.

B. DESIGN DATABASE

1. Informal Solution

The DDB must contain a class structure and methods which support the storage, retrieval, and management of CAPS prototypes. Next we integrate the ONTOS object oriented Database Management System with locally engineered classes and methods designed to support the informal solution and stated requirements. This equates to designing a database schema. Choosing this approach carries several distinct advantages:

- Saves money because the DDB can be developed in less time
- The ONTOS system provides an extensive developer's library
- Enhances Security
- Network capabilities are built in
- Encourages the use of off the shelf generic software programs
- Incorporates enhanced documentation on data structures provided by ONTOS
- Ensures modularity

These and other advantages are discussed in Chapter II.

2. Entities and Their Relationships

The set of files reflected in Figure 8 become the primary attributes of a versioned component in the database. Other attributes required to effectively scope and manage these components and the unique capabilities supported by versioning are contained in the ER diagram shown in Figure 12.

Figure 13 shows the lower level entities required to support the data structure of a versioned component. The entities THREAD, COMPONENT, and TEXT_OBJECT reflect the core elements stored as objects in the ONTOS database.

3. Pre-defined ONTOS Classes

The main ONTOS classes used in this DDB application are:

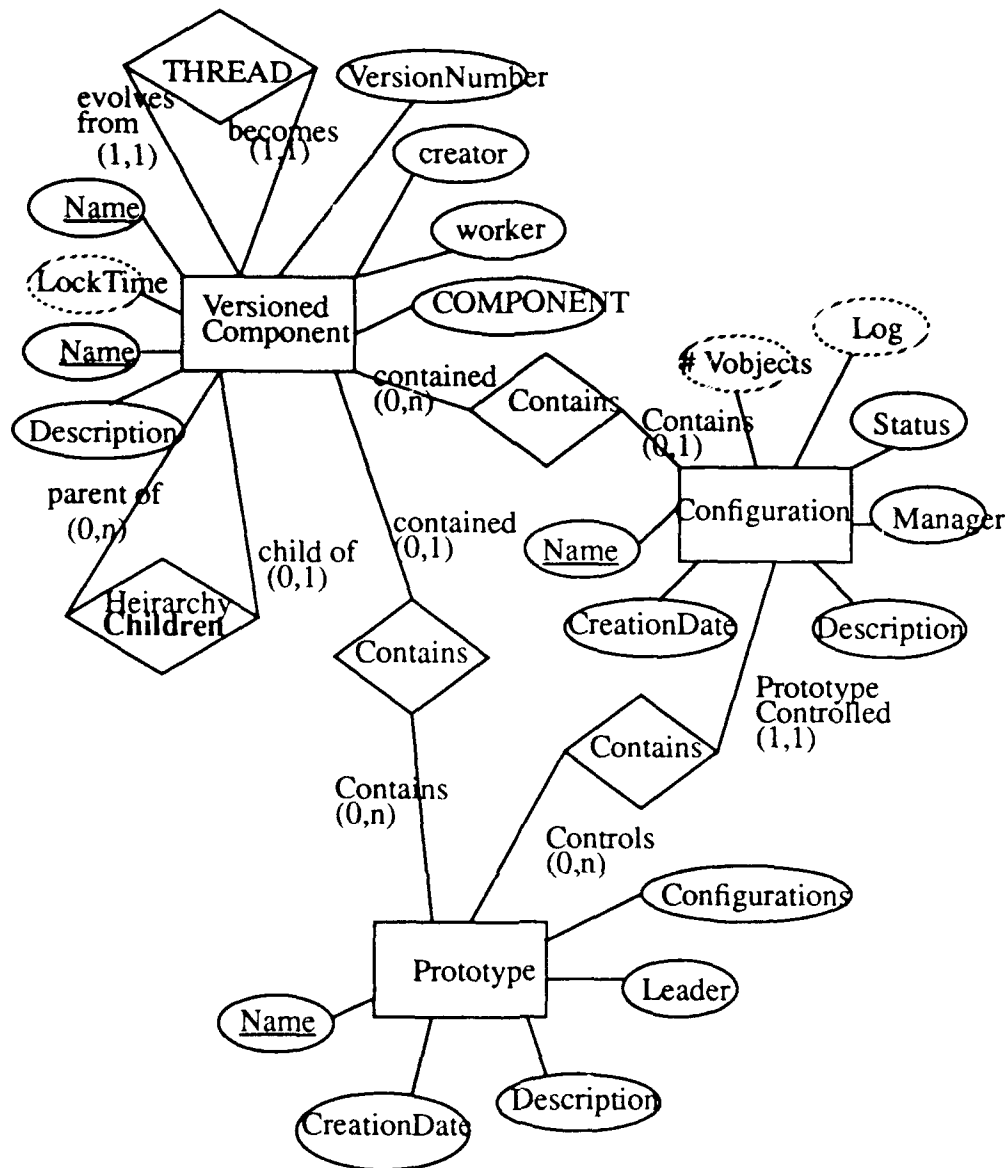
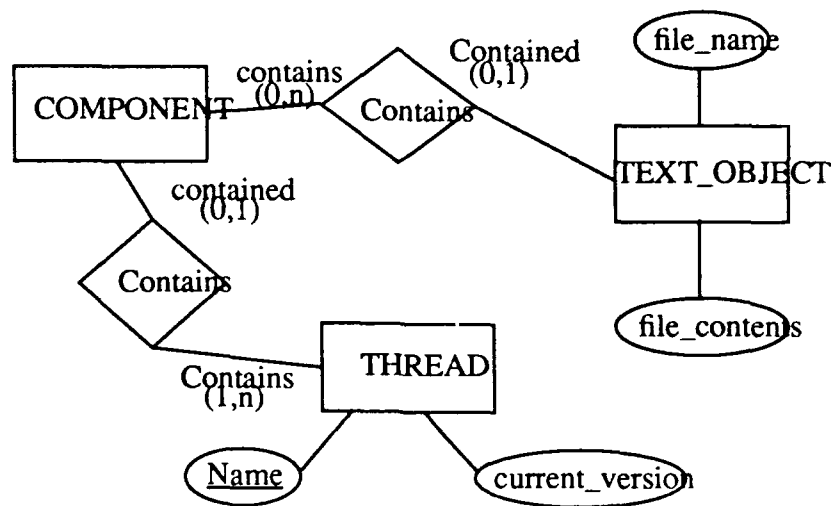


Figure 12. Design Database Entity Relationship Diagram

a. Object

Object is the class used to create persistent objects in the DBMS. All classes requiring persistence must inherit from Object. Persistent objects exist for longer than the immediate I/O session. An object is persistent if you can store it in the database and retrieve it at a later time. If an object is persistent it must have a unique id. This identification



number may be explicitly assigned by the DDB application or defaulted to a unique id key field generated by methods provided by this ONTOS class. This allows us to store several different components in the database with the appearance of each component having the same name.

b. List

Lists belong to a container class which can store other objects of any type. As a basic data structure, its sister class `List_Iterator` is used to iterate from one instance of an object to the next.

c. Dictionary

Working on an indexed set of objects contained within it, Dictionary is also a container class. It differs from List in its implementation and in its available methods for directly accessing any item contained within it based on key values.

4. Design Database Classes

The DDB classes and methods work to provide the user the functionality of a library. You can check components in or out. However, unlike a library, when you check

components out of the database for updating, a “copy” of the components is put into your private work area. These components are then flagged inside the DDB as being “locked”. This lock prevents other designers from checking out the same components for updating. The original component remains in the database and can never be altered. It is an immutable copy of the designer’s prototyping efforts at the time it was checked into the DDB. Other workers who attempt to check out the same versioned components may “view” the prototype, but may not modify or evolve the prototype into a newer version. The designer unlocks the prototype when it is checked back into the database. If changes were made to a component then that component will version. If no changes were made then versioning will not be necessary and will not occur. In either case, when a prototype is checked back into the DDB, the locks on components in that prototype are removed.

A diagram of the class hierarchy for the persistent objects in the DDB is included as Figure 14. A class is a high level abstraction that represents something in the real world. All classes developed to support this model are further elaborated with a class description, behavior, attributes, and operations in the rest of this section:

Persistent DDB Classes

Class COMPONENT

COMPONENT is an abstract class in the DDB. It is the core element in the DDB and represents a composite or atomic PSDL operator/type. As reflected in the ER diagram contained in Figure 10, a component contains a variable number of TEXT_OBJECTS.

COMPONENT, like all persistent classes, is a descendant of the ONTOS class **Object**. If a class intends to store persistent objects in the ONTOS database then the class must inherit from the Object class.

A diagram of these dependencies is shown in Figure 14.

Behavior:

COMPONENT is an abstract container class for TEXT_OBJECTS. It is important to note the distinction between atomic and composite components in the DDB. Atomic objects differ from composite objects in the following way: composite objects

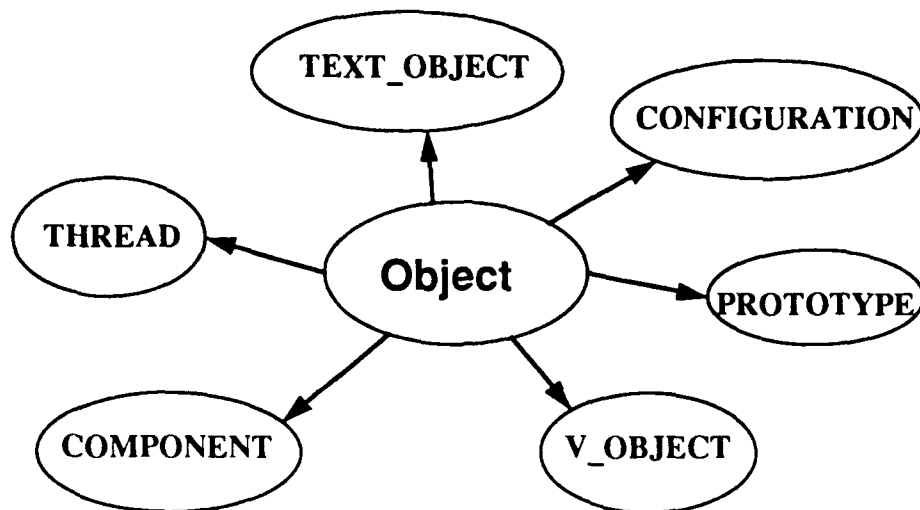


Figure 14. Design Database Descendants of Class Object

decompose into other composite and/or atomic objects, but an atomic object has no further decompositions. In the multi-way tree utilized by this application, atomic objects have no children.

Set of Attributes:

`text_object_list` -> contains the list of text_objects in the component. May contain between zero and five of the following files:

Postscript: A file containing Postscript Description Language required by the graphic editor.

Graph: A file containing shapes and geometric point information required by the graphic editor.

Implementation: A file containing either Ada source code or a PSDL decomposition. This file is used by the syntax directed editor, the dynamic scheduler, and the translator.

Specification: A file containing Prototype Specification Description Language (PSDL). This file is used by the syntax directed editor, the software base, the dynamic scheduler, and translator.

Source: ADA Code - a special file containing the source code for the component and/or prototype.

Operations:

COMPONENT -> creates an instance of the COMPONENT class.

getDirectType -> returns an ONTOS Type¹.

getComponentNames -> returns the names of the TEXT_OBJECTS.

getComponentSource -> restores the Postscript, graph, implementation, specification, and source file attributes of a versioned component to the appropriate files in the designer's work area.

addTextObject -> adds a TEXT_OBJECT to the COMPONENT.

getPSfile -> restores the Postscript file attribute of a versioned component to the appropriate files in the designer's work area.

getGRAPHfile -> restores the graph file attribute of a versioned component to the appropriate files in the designer's work area.

getSPECfile -> restores the graph file attribute of a versioned component to the appropriate file in the designer's work area.

getIMPfile -> restores the graph file attribute of a versioned component to the appropriate file in the designer's work area.

getSOURCEfile -> restores the graph file attribute of a versioned component to the appropriate file in the designer's work area.

Class CONFIGURATION

When a large prototype is developed and evolved from one version to another, the user may wish to selectively group different versions of the component and its graph subtree into more manageable sets. CONFIGURATION is an abstract class which allows the designer to more effectively deal with selected subsets of components. Remembering that the database will eventually consist of several versions of a prototype and its

1. Refer to Chapter II Section C for definition of ONTOS Type

decomposed components, the user may decide to set up a configuration to reflect version number 1 (CONFIG1) and a newer configuration to reflect those components contained in version number 2 (CONFIG2). Working with configurations, the user may decide that the earlier configuration needs further refinement or that the older configuration is more suited to his application development needs. The DDB will allow the prototype developer to check out the older configuration and retrieve all components contained in that configuration with one single command. When the DDB is fully matured, that configuration will be checked back into the database as version 1.1 - spinning off a new thread from the base thread. This concept of a variation is not implemented in this version of the DDB, but will be developed in the next generation of the tool.

Behavior:

CONFIGURATIONS are entered into the database with commands given in Appendix B. A versioned component is then attached to the configuration as that configurations' default versioned object. With this hook into the prototype tree structure, configurations allow the designer to navigate the prototype hierarchy - to store and/or retrieve versioned components quickly and efficiently to different work areas.

Set of Attributes:

config_status -> reflects the current status of design work. Possible values are 'A' - active; 'T' - archive to tape.

config_manager -> shows who is managing the configuration.

ConfCreationDate -> time/date that the configuration was created.

conf_num_vobjects -> derived field containing the number of versioned components in the configuration.

config_log_entry -> contains a running, non-erasable log journal reflecting all changes to the configuration.

config_description -> A modifiable attribute which contains a generic description of the configuration.

theVersioned_Object -> the default versioned component which serves as this configuration's entry point to the prototype hierarchical data structure. There is only one default versioned component per configuration. This is attached to a configuration after the configuration is inserted into the DDB.

Operations:

CONFIGURATION -> creates an instance of the CONFIGURATION class.

getDirectType -> returns an ONTOS Type.

Destroy -> cleans up the heap.

getConfigName -> displays the configuration name to stdout.

name -> returns the configuration name.

getConfigStatus -> displays the configuration status to stdout.

getConfigManager -> displays the configuration manager to stdout.

getConfigLog -> displays the configuration Log to stdout.

getConfigDescription -> displays the configuration description to stdout.

dumpConfigSummary -> displays the date created, manager, default versioned component, and description to stdout.

listConfigOperators -> lists the entire configuration subtree (all operators from default versioned component).

updateConfigManager -> updates the manager attribute.

updateConfigName -> updates the name attribute.

updateConfigStatus -> updates the status attribute.

addToConfigLog-> appends entry to the log attribute.

updateConfigDescription -> updates the configuration description.

Class PROTOTYPE

PROTOTYPE is the highest level abstract class. A prototype represents all of the information associated with one prototyping project.

Behavior:

PROTOTYPES are entered into the DDB with commands reflected in Appendix B. PROTOTYPE is a container class which maintains references to CONFIGURATIONS and versioned components (V_OBJECTS). This ability to logically group several configurations and/or components into one prototype allows the designer to more easily manage a variable number of projects at any one time.

Set of Attributes:

protleader -> the name of the prototyping project leader.

protDictIndex -> Required by ONTOS for Dictionary maintenance.

protCreationDate -> The time/date that the prototype was created.

protDescription -> A reference to a TEXT_OBJECT containing a text description of the prototype.

prot_configuration_list -> A reference to a list of configurations associated with a prototype.

prot_default_configuration -> A reference to the last configuration inserted into the prototype.

Operations:

PROTOTYPE -> creates an instance of the PROTOTYPE class.

getDirectType -> returns an ONTOS Type.

Destroy -> cleans up the heap.

getName -> returns the PROTOTYPE object name.

getConfigName -> returns default configuration object name.

getPrototypeName -> displays prototype name to stdout.

getPrototypeLeader -> displays prototype leader to stdout.

getPrototypeDescription -> displays prototype description to stdout.

changePrototypeName -> substitutes new name for existing name.

changePrototypeLeader -> substitutes new leader for existing leader.

updatePrototypeDescription -> substitutes new description for current description.

dumpPrototypeSummary -> displays the date created, leader, default configuration name, and description to stdout.

addConfiguration -> adds configuration to prototype configuration list.

listConfigurations -> lists configurations to stdout.

setProtCreationDate -> sets the prototype creation date to system time.

getProtCreationDate -> returns a time/date of prototype creation.

getDefaultConfigName -> displays the default configuration name to stdout.

getConfiguration -> returns the configuration requested.

getDefaultConfiguration -> returns the default configuration.

getVobject -> returns the most current versioned component matching the prototype name.

Class TEXT_OBJECT

TEXT_OBJECT contains a file name and its associated text.

Behavior :

Accept the file name, construct an object in the ONTOS database to store the name and file contents, and read the file contents into the persistent object.

Set of Attributes:

the_file_name -> name of the tool file.

the_text -> contents of the text file.

Operations:

TEXT_OBJECT -> creates an instance of the TEXT_OBJECT class.

getDirectType -> returns an ONTOS Type.

Destroy -> cleans up the heap.

append -> reads the contents of the file into the ONTOS object.

text -> sends a stream to output file.

rebuildTextFile -> rebuilds the text file in the user's work area and returns completion status.

displayFileName -> displays the file name to stdout.

getFileName -> return the file name.

text -> returns the object's text.

resetTheText -> resets the text attribute.

Class THREAD

THREAD is a high level abstract class which contains one or more versions of the same component. Additional components added to a thread have a version number determined by incrementing the highest version number of a component in the thread by one.

Behavior :

A thread is created for every new component entered into the database. Most methods check for the existence of a THREAD prior to executing a requested operation.

Set of Attributes:

current_version -> version number of the most recent version on the thread.

the_list -> an ONTOS reference to a List containing different versions of a component.

Operations:

THREAD -> creates an instance of the THREAD class.

getDirectType -> returns an ONTOS Type.

Destroy -> cleans up the heap.

getCurrentVersionNum -> returns the integer current version.

current -> returns the versioned component (V_OBJECT) having the same version number as the current version attribute of the thread.

version -> returns the versioned component (V_OBJECT) having the version number supplied by the calling method.

add_object -> adds a new versioned component (V_OBJECT) to the thread.

displayThreadVersions -> display the version number of every versioned component (V_OBJECT) in the thread.

displayThreadContents -> displays the version number and the description of every versioned component (V_OBJECT) in the thread.

Class V_OBJECT

V_OBJECT is an abstract persistent class. It represents an immutable snapshot in time of an operator/type that was checked into the database as part of a prototype design.

Behavior:

V_OBJECTS are entered into the DDB with commands reflected in Appendix B. Only the root V_OBJECT must be entered by the user (tool interface). The TREE class then scans the directory for other operators which are descendants of that operator and constructs a tree structure for comparison to the contents of the database.

Set of Attributes:

theVersionNumber -> the component's version number.

creationDate -> the date the component was created.

lockTime -> the date the component was created. Evaluated as the system's epoch time if not locked. Epoch time means no lock is set and lockTime equals zero.

node_name -> short node name. No ancestor information. The long name is contained in the THREAD class.

creator -> name of the end user who created the component.

worker -> if the component is locked, then this attribute contains the name of the worker who checked it out.

visited -> a boolean used to navigate the tree structure.

last_op_checkin -> a boolean used to prevent duplicate checkin operations.

theDescriptionPtr -> TEXT_OBJECT containing the description for this versioned component.

theThreadPtr -> pointer to the THREAD this versioned component is contained in.

theCOMPONENTPtr -> points to the component which contains the text object's for this versioned object.

theChildPtr -> points to a list of Children.

theParentPtr -> points to the most current parent.

Operations:

V_OBJECT -> creates an instance of the V_OBJECT class.

getDirectType -> returns an ONTOS Type.

Destroy -> cleans up the heap.

connect_vobject_to_thread -> attaches a versioned component to a thread.

setParent -> set's versioned components Parent.

setNodeName -> pulls the short node name from the thread name.

getNodeName -> returns the short node name.

getVobjName -> displays the component name.

getName -> returns the component name.

resetVisitedFlag -> used to navigate the tree.

setVisitedFlag -> used to navigate tree.

getVisitedFlag -> TRUE if visited, FALSE otherwise.

getVObjComponentsName -> display the names of the text objects associated with this versioned component.

displayVersionNumber -> displays the component's version number to stdout.

getVersionNumber -> return the component's version number.

dumpVobjSummary -> displays the date created, creator, worker (or None if unlocked), lockTime (or NONE if unlocked), and description to stdout.

setCreationDate -> The time/date that the component was created

getCreationDate -> returns the time/date the component was created.

setLock -> set's the lockTime immediately following checkout.

getWorker -> return the worker who has the component locked.

getCreator -> return the end user who created the component into the DDB.

setWorker -> set the worker to the \$USER environment variable.

resetLastOpTrue -> if last operation was a checkin, then set the last operation checkin attribute to TRUE. prevents duplicate checkins.

resetLastOpFalse -> on a checkout, reset the last operation checkin to FALSE.

get_last_operation -> returns TRUE if last operation was a checkin.

releaseLock -> resets lock and returns completion status.

getLockTime -> returns the lockTime of the component.

getDescription -> displays the description to stdout.

listChildren -> displays a list of children of this versioned component to stdout.

longlistOperatorNames -> displays a long list of all operators/types in the subtree of this versioned component.

listOperatorNames -> displays a list of all operators/types in the subtree of this versioned component.

updateDescription -> updates the description attribute.

addComponentNode -> adds a component to this versioned object.

deleteChildNode -> deletes a reference from the list theChildPtr.

addChildNode -> adds a reference to the list theChildPtr.

getParent -> returns the parent of this versioned component.

getCOMPONENT -> returns the component of this versioned object.

dumpSubtree -> rebuilds the versioned components in read-only or read-write mode into an end user's \$PROTOTYPE directory.

releaseLockSubtree -> releases the lockTime on all versioned components in this component's subtree.

getChildren -> returns the list of Children of this versioned component.

getChildPtr -> returns TRUE if the cardinality of children of this component is greater than zero.

checkoutCOMPONENTNode -> rebuilds all text objects of the component attribute of this versioned object and returns completion status.

Non Persistent DDB Classes

Class DIRECTORY

A non-persistent class designed to evaluate the status of prototypes being checked into the DDB.

Behavior:

This class has no direct relationship to the end user interface. The directory class searches the \$PROTOTYPE subdirectory of the designer, determines what operators exist that match the prototype name, and develops a linked list of nodes to be inserted into a multi-way tree. The output from this class is passed to the TREENODE class for insertion of nodes into the persistent database.

Set of Attributes:

TREENODE_linked_list -> operator/type nodes contained in the user's PROTOTYPE directory.

Operations:

DIRECTORY -> constructor for building a directory scanner.

read_directory -> Given a root operator, scans the user subdirectory for prototype component matches. Builds a list of all prototype components. Stores the potential component matches in a linked list of operator nodes.

updatetimestamp -> evaluates whether component files in the user work area are newer than the versioned component stored in the ONTOS database. This information is used by the TREENODE class when determining whether to create a new version of the component.

find_treenode -> scans the linked list of operator nodes to build the multilevel k-ary tree of TREENODES. Both operator nodes and TREENODES use the same class structure, except that the operator nodes have no children (subtree) references.

getOperatorList -> required to build the multilevel k-ary tree. Returns the operator list defined by evaluating the users work area with the read_directory operation.

Class QUEUE

QUEUE contains the basic non-persistent queue data structure required to construct the multi-level k-ary hierarchy.

Behavior:

Implements basic queue functions.

Set of Attributes:

None.

Operations:

put -> append an operator TREENODE on the queue.

get -> get an operator TREENODE from the queue.

empty -> test the queue for empty status.

Class slist, slink, slist_iterator, slink_iterator

These classes are non persistent abstract classes providing the generic capabilities of a singly linked list.

Behavior :

these classes contain the basic data structures required for queues and linked lists.

Set of Attributes:

e -> treenode pointer.

next -> link pointer.

ce, last -> slink pointer.

cs -> slist pointer.

Operations:

slist, slink, slist_iterator, slink_iterator -> constructors.

insert -> add TREENODE at head of list.

append -> add TREENODE at tail of list.

get -> return TREENODE.

clear -> remove all links.

empty -> test for empty list.

Class TREE

A non-persistent class designed to build a multi-way tree of operator/types developed by the end user in the \$PROTOTYPE directory.

Behavior:

The TREE class will establish the root operator and build the tree.

Set of Attributes:

tree_name -> assumes the name of the prototype root versioned component.

theTreeRootNode -> TREENODE (root of multi-level k-ary tree).

Operations:

TREE -> constructor for building the root TREENODE.

build_tree -> builds the multi-level tree.

find_treenode -> returns a TREENODE from the input list of Operator nodes returned from DIRECTORY.

Class TREENODE

The most important non-persistent class designed to build a multi-way tree of operator/types developed by the end user in the \$PROTOTYPE directory.

Behavior :

The TREENODE class is used to establish the operator node list, to convert the operator node list to a multi-level tree, and to compare each node in the multi-level tree to versioned components in the database. Component versioning occurs where the TREENODES have newer timestamps than the lockTime on the versioned components.

Set of Attributes:

`tree_node_name` -> explicit name of node (from root node to level in subtree). Taken in its entirety, the `tree_node_name` contains the sequence of ancestor names from the entire prototype name to the given sub-component's `node_name`.
`node_name` -> unique name of operator/type without level/ancestor information.
`timestamp` -> contains the operator/types most recent system time. Each tool updates the file when it creates or updates its product files.
`level` -> root operator equals 1. Other nodes reflect the level where they reside.
`ChildrenList` -> linked list of `TREENODE` descendants of this instance.
`ParentNode` -> parent `TREENODE` of this instance.

Operations:

`TREENODE` -> constructor which reads directory and builds operator list.
`updatetimestamp` -> reads the directory again for the newest of potential files for each operator/type developed by CAPS tools. Stores the result in `timestamp`.
`getname` -> returns the `TREENODE` name.
`insertChildNode` -> inserts a `TREENODE` into the `ChildrenList`.
`getChildren` -> returns the linked list of `Children TREENODES`.
`getParentNode` -> returns the `Parent TREENODE` of this instance.
`get_asc_time` -> returns the 26 character ASCII time from the standard C function `ctime`.
`getlevel` -> return the level of this `TREENODE`. root is level 1, successive levels increment by 1.
`get_long_time` -> return the `timestamp`.
`list_subtree` -> lists each `TREENODE` from the root to each leaf node.
`checkin_subtree` -> initiates the recursion for comparing the tree of `TREENODES` to the hierarchical structure in the ONTOS database.
`checkin_node` -> does the bulk of the loading into the ONTOS database. Creates new threads and versioned components when the `TREENODE timestamp` is more

current than the versioned component's lockTime or where no thread with that operator name exists (new operator).

C. TESTING AND EVALUATION

The method used for checking the functionality of the DDB required an exhaustive check of many possible variations of the functions documented in Appendix B. To demonstrate that each of the commands produce the desired output, a unix shell script was generated which tested all of the functions. The complete version of this shell script test is contained in Appendix C.

The output from this sequence of commands, also contained in Appendix C, shows that the testing phase answered two very important questions affirmatively - that this tool could successfully store and retrieve versioned components of prototypes as dictated by the design criteria; and that the software engineering design database could be developed as a stand-alone tool which interacted with other tools through the tool interface.

The results of some commands can not be verified by running that command alone. It is only by running several commands in a series that the effect of one command changing the state of the database can be verified. Consequently, these types of tests were included in the testing phase.

D. MAINTENANCE NOTES

All code generated by the authors in support of this implementation effort is contained in Appendix D.

This code was developed using version 2.0 of the Glockenspiel C++ compiler and the beta version 2.01 of the ONTOS object-oriented Database Management System. ONTOS tools included packages containing extensive capabilities for debugging in the form of the ONTOS graphical database browser (DBrowser).

Special ONTOS header files and the makefiles used during compilation are contained in Appendix E.

V. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

The goal of this thesis research was to design and develop an engineering database which could store, version, and retrieve software prototype components. This database provides concurrent users the command functions to insert prototypes, store and manage configurations, and insert and retrieve components throughout the software engineering life cycle.

The prototype functions allow the designer to store and retrieve management information concerning any of several potential prototype designs.

The configuration functions allow the designer to more easily navigate the various versions of a software prototype. It provides him a "hook" for navigating the hierarchy from any specified component. With additional features such as grouping by configuration and logging, configurations provide powerful features and a sound design for the development of evolution management into the engineering database model.

The versioned component functions give the designer the flexibility to checkout for modification or viewing specific prototype components; to expand, modify, and decompose these components with other tools in the CAPS system; and to check selected portions of those components back into the database at any time[Ref. 19].

B. RECOMMENDATIONS FOR FUTURE WORK

1. Variations

Variations will allow users to check out older versions of components and/or groups of components and evolve them to newer versions. When the updates are completed components will version correctly - splitting, updating, and rejoining threads where appropriate. Work must be done on other tools in the CAPS system to make them accept as input and produce as output components which contain version information. This will

provide the DDB with the necessary information for re-attaching components to the hierarchical structure at the correct location.

2. Security

This implementation of the engineering database does not address the issue of security other than the concurrency control mechanisms. This thesis makes no effort to distinguish between different levels of access or authorization for access to objects and their attributes. This is a subject that requires additional research and development.

3. Evolution Management

This program must be enhanced to incorporate the "uses" and "derives" properties [Ref. 11]. The database will then be capable of automatically flagging the effects that changing one component has on other components in the prototype. As components are checked out of the database they must take information with them pertaining to the version checked out. This is required to reattach components to the multi-way tree upon update. These and other evolution attributes will become an important ingredient for providing a fully functional engineering database.

APPENDIX A

ENVIRONMENTAL MODEL

A. STATEMENT OF PURPOSE

The Purpose of the design database is to provide the designer a method of storing and retrieving components generated by the various tools of the COMPUTER AIDED PROTOTYPING SYSTEM (CAPS). This tool shall be flexible enough to expand as other tools are developed. It shall be capable of maintaining older versions of components as new components are developed. It shall allow the user to version new and/or older components with the common interface considerations.

B. EVENT LIST

List prototypes

List current version of prototype

Select current version of prototype

Check out current version of prototype's components for viewing/update

Add new version of prototype's components

List previous versions of prototype

Select previous version of prototype

Check out previous version of prototype's components for viewing/update

List selected prototype attributes (creation date, leader, etc)

Update selected prototype attributes (leader, description, etc)

List composite components (operators/types)

List atomic components (operators/types)

List selected versioned component attributes (date created, description, etc.)

Update selected versioned component attributes (date created, description, etc.)

List configurations

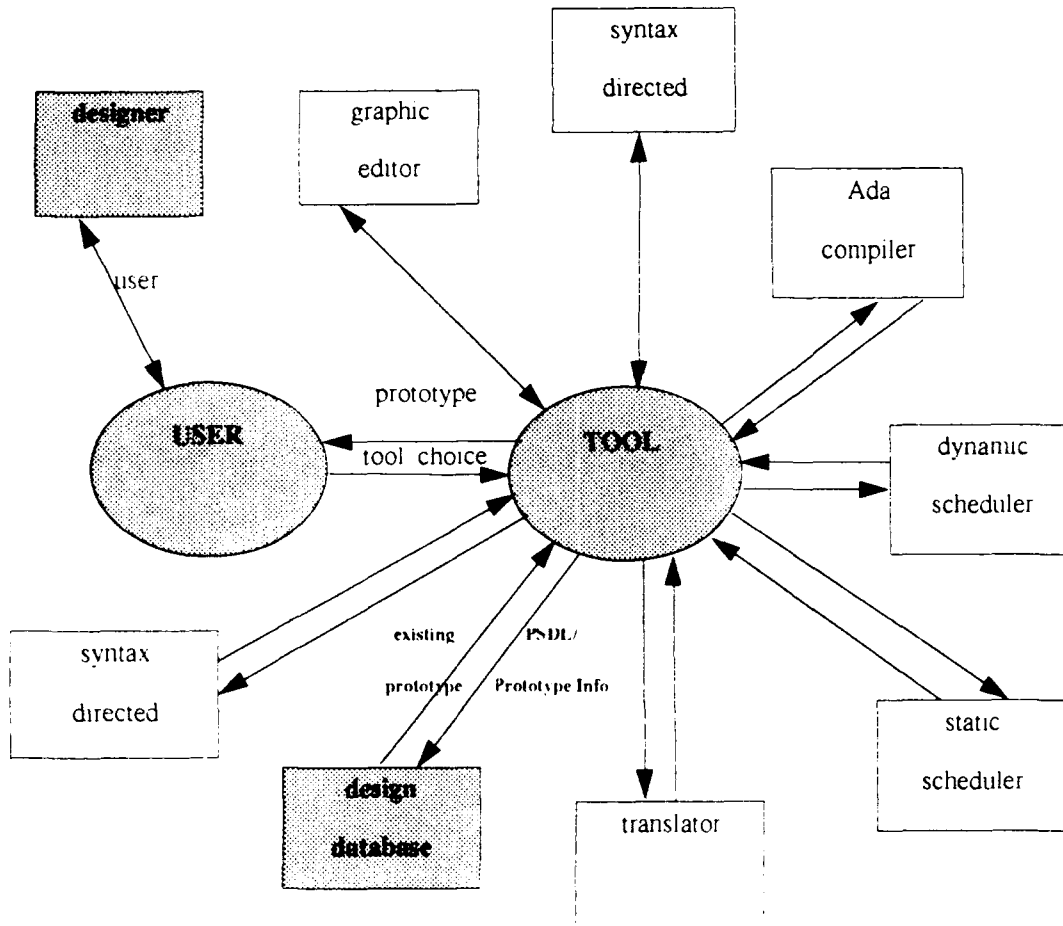
List current default version of configuration in prototype

Select current default version of configuration in prototype

Check out current version of configuration for update

Add new version of configuration's components

C. CONTEXT DIAGRAM



CAPS Interface Data Flow Diagram (partial/modified) [Ref. 3:p. 19]

APPENDIX B

COMMAND INTERFACE

A. PROTOTYPE COMMANDS

Table 1: PROTOTYPE COMMANDS

Tag	Description	Output	Call ^a
PIP	Insert Prototype	Confirmation	ddb <db> pip <p> ^b [leader] [description] ^c
PLN	List Names	Names (1 per line)	ddb <db> pln
PLL	Long List	Name/Default Config/ Default VOBJECT	ddb <db> pll
	<i>Retrieval Commands:</i>		
PDS	Dump Summary	Date Created Leader Default Config Description	ddb <db> pds <p>
PRD	Retrieve Date	Date Created	ddb <db> prd <p>
PGL	Get Leader	Leader	ddb <db> pgl <p>
PGC	Get Configuration	Default Config	ddb <db> pgc <p>
PGD	Get Description	Description	ddb <db> pgd <p>
	<i>Update Commands:</i>		
PUL	Update Leader	Confirmation	ddb <db> pul <p> "new leader"
PUD	Update Description	Confirmation	ddb <db> pud <p> file

a. <angle brackets> reflect required parameters. [square brackets] denote optional parameters.

b. <db> db=any Ontos Database name. <p> p=any prototype name

c. Passed in as a file-name or string

B. CONFIGURATION COMMANDS

Table 2: CONFIGURATION COMMANDS

Tag	Description	Output	Call ^a
CIC	Insert Configuration	Confirmation	ddb <db>cic <p> <c> ^b [manager] [description] ^c
CLN	List Names	Names (1 per line)	ddb <db> cln <p> <c>
CLV	List Default VOBJECT	Name/Version	ddb <db> clv <p> <c>
CLO	List Operators	Operator Name ^d / Version	ddb <db> clo <p> <c>
CLL	Long List Default VOB- JECT's Children	Node ^e Name/Ver- sion	ddb <db> cll <p> <c>
	<i>Retrieval Commands:</i>		
CDS	Dump Summary	Date Created Date Changed Manager Default VOBJECT/ Version Description	ddb <db> cds <p> <c>
CDA	Get Date Created	Date Created	ddb <db> cda <p> <c>
CGM	Get Manager	Manager	ddb <db> cgm <p> <c>
CGD	Get Description	Description	ddb <db> cgd <p> <c>
CGL	ViewLog ^f	Log Entries	ddb <db> cgl <p> <c>
	<i>Update Commands:</i>		
CUN	Update Name	Confirmation	ddb <db> cun <p> <c> new_name
CUM	Update Manager	Confirmation	ddb <db> cum <p> <c> new_name
CUD	Update Description	Confirmation	ddb <db> cud <p> <c> file

Table 2: CONFIGURATION COMMANDS

Tag	Description	Output	Call ^a
CPL	Post Log	Confirmation	ddb <db> cpl <p> <c> file
CRL	Release Lock	Confirmation	ddb <db> crl <p> <c> [version]
CAA	Update VOBJECT's sub- tree	Confirmation	ddb <db> caa <p> <c>
CAO	Attach VOBJECT to CON- FIGURATION	Confirmation	ddb <db> cao <p> <c> <vobject> [version]
	<i>Extraction Commands:</i>		
CDT	Dump VOBJECT subtree	file(s)	ddb <db> cdt <p> <c> R/W ^g [version]

a. <angle brackets> reflect required parameters. [square brackets] denote optional parameters.

b. <db> db=any Ontos Database name. <p> p=any prototype name <c> c = any configuration name

c. Passed in as a file-name or string

d. Operator name containing explicit path information for determining location (level) in heirarchical data structure

e. Node name containing no reference to location (level) in heirarchical data structure

f. Log may be viewed only. Log can not be updated. Additional log entries are posted to the bottom of the current log.

g. R/W dump file for Read-Only (R) or Write (W). Tool interface must supply either R or W to build files.

C. VERSIONED COMPONENT COMMANDS

Table 3: VERSIONED COMPONENT COMMANDS

Tag	Description	Output	Call ^a
VAA	Add VOBJECT & Sub-tree	Confirmation	ddb <db>vaa <p> <v> ^b
VLO	List Operators	VOBJECT Name ^c / Version	ddb <db> vlo <p> <v> [version]
VLL	Long List VOBJECT's Children	VOBJECT Name ^d / Version	ddb <db> vll <p> <v> [version]
VLP	Long List VOBJECT's Parent & Siblings	VOBJECT Name/Ver- sion	ddb <db> vlp <p> <v> [version]
	<i>Retrieval Commands</i>		
VDS	Dump Summary	Date Created Creator Worker (or NONE) Lock Time (or NONE) Description	ddb <db> vds <p> <v> [version]
VDD	Get Date Created	Date Created	ddb <db> vdd <p> <v> [version]
VGL	Get Lock Time	Date/Time Locked	ddb <db> vgl <p> <v> [version]
VGv	Get Versions	Version Number (1 per line)	ddb <db> vgv <p> <v>
VGD	Get Description	Description	ddb <db> vgd <p> <v> [version]
	<i>Update Commands:</i>		
VUD	Update Description	Confirmation ^e	ddh <db> vgd <p> <v> file ^f [version]

Table 3: VERSIONED COMPONENT COMMANDS

Tag	Description	Output	Call ^a
VAA	Add VOBJECT and subtree	Confirmation	ddb <db> vaa <p> <v> [version]
VRO	Release Lock Operator	Confirmation	ddb <db> vro <p> <v> [version]
VRS	Release Lock Operator and Subtree	Confirmation	ddb <db> vrs <p> <v> [version]
<i>Extraction Commands:</i>			
VGP	Get Postscript	file	ddb <db> vgp <p> <v> R/W ^g [version]
VGG	Get Graphics	file	ddb <db> vgg <p> <v> R/W [version]
VGI	Get Implementation	file	ddb <db> vgi <p> <v> R/W [version]
VGC	Get Specification	file	ddb <db> vgc <p> <v> R/W [version]
VGS	Get Source	file	ddb <db> vgs <p> <v> R/W [version]
VDF	Dump Operator Atomic ^h	file	ddb <db> vdf <p> <v> R/W [version]
VDT	Dump Operator and Subtree	file(s)	ddb <db> vdt <p> <v> R/W [version]

- a. <angle brackets> reflect required parameters. [square brackets] denote optional parameters.
b. <db> db=any Ontos Database name. <p> p=any prototype name <c> v=any vobject name
c. Operator name containing explicit path information for determining location (level) in heirarchical data structure
d. Node name containing no reference to location (level) in heirarchical data structure
e. Only Creator may update description

- f. file containing description
- g. R/W dump file for Read-Only (R) or Write (W). Tool interface must supply either R or W to build files
- h. Atomic is considered collection containing .ps .graph .imp.psd1 .spec.psd1 and .a file(s)

APPENDIX C

TESTING AND EVALUATION

A. NOTES ON TESTING AND EVALUATION

For functions which return no visible output, the successful completion is marked by the standard output of:

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Three Prototypes are inserted into the Engineering Database. These include one Command and Control Application (c3i) and two classroom applications (FishFarm and Robot). The purpose for inserting three applications is to test working with several prototypes. After demonstrating these functions, the remaining functions are run against the more comprehensive prototype created by the CAPS system called c3i. This prototype was selected due to its ability to fully exercise the "graph walking" capabilities of the model.

B. TEST SCRIPT

```
#!/usr/local/bin/tcsh
clear
echo "*****"
echo " * NAVAL POSTGRADUATE SCHOOL *"
echo " * Software Engineering *"
echo " * Design Database *"
echo " * Thesis Advisor Dr. Luqi"
echo " * Written by: Drew Dwyer and Garry Lewis *"
echo " * Design Database v1.1 WP.D.A.L.G . *"
echo "*****"
echo ""
echo ""
echo -n "Press Return to begin demonstration "
set a=$(S<)
clear

echo "Running main ddb53 pip c3i ..."
echo ""
main ddb53 pip c3i
echo "Running main ddb53 pip fishfarm 'Drew Dwyer' ..."
echo ""
main ddb53 pip fishfarm "Drew Dwyer"
echo "Running main ddb53 pip robot 'Garry W. Lewis' file1 ..."
echo ""
main ddb53 pip robot "Garry W. Lewis" file1

echo ""
```

```
echo -n "INSERT PROTOTYPE test complete ... Press Return to continue "  
set a=($<)  
echo ""  
clear
```

```
echo "Running main ddb53 pgd robot ...returns description to screen..."  
echo ""  
main ddb53 pgd robot  
echo "Running main ddb53 pud robot file2 ...updates description ..."  
echo ""  
main ddb53 pud robot file2  
echo "Running main ddb53 pgd robot ...returns the new description."  
echo ""  
main ddb53 pgd robot  
echo ""  
echo -n "GET/UPDATE PROTOTYPE DESCRIPTION test complete ... Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 prd c3i "  
echo ""  
main ddb53 prd c3i  
echo ""  
echo -n "RETRIEVE PROTOTYPE DATE test complete ... Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 pul robot Larry Williamson ..."  
main ddb53 pul robot "Larry Williamson"  
echo ""  
echo ""  
echo ""  
echo "Running main ddb53 pgl robot"  
main ddb53 pgl robot  
echo ""  
echo -n "UPDATE/GET PROTOTYPE test complete ... Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 pds c3i ..."  
echo ""  
main ddb53 pds c3i  
echo -n "Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 pds fishfarm ..."  
echo ""  
main ddb53 pds fishfarm  
echo -n "Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 pds robot ..."  
echo ""  
main ddb53 pds robot
```

```
echo ""
echo -n "DUMP PROTOTYPE SUMMARY test complete ... Press Return to continue "
set a=$(S<)
clear
```

```
echo ""
echo "Running main ddb53 pln ..."
echo ""
main ddb53 pln
echo ""
echo -n "LIST PROTOTYPE NAMES test complete ... Press Return to continue "
set a=$(S<)
clear
```

```
echo "Running main ddb53 cic c3i unix ..."
echo ""
main ddb53 cic c3i unix
echo ""
echo "Running main ddb53 cic c3i dos 'Drew Dwyer' ..."
main ddb53 cic c3i dos "Drew Dwyer"
echo ""
echo "Running main ddb53 cic c3i sparc 'Garry Lewis' file1 ..."
main ddb53 cic c3i sparc "Garry Lewis" file1
echo ""
echo -n "INSERT CONFIGURATION test complete... Press Return to continue "
set a=$(S<)
clear
```

```
echo "Running main ddb53 cum c3i dos 'Marty Shoppenheimer' ..."
main ddb53 cum c3i dos "Marty Shoppenheimer"
echo ""
echo -n "UPDATE CONFIGURATION MANAGER test complete... Press Return to continue "
set a=$(S<)
clear
```

```
echo "Running main ddb53 cds c3i unix ..."
main ddb53 cds c3i unix
echo -n "Press Return to continue "
set a=$(S<)
clear
```

```
echo "Running main ddb53 cds c3i dos ..."
main ddb53 cds c3i dos
echo -n "Press Return to continue "
set a=$(S<)
clear
```

```
echo "Running main ddb53 cds c3i sparc ..."
main ddb53 cds c3i sparc
echo -n "DUMP CONFIGURATION SUMMARY test complete... Press Return to continue "
set a=$(S<)
clear
```

```
echo "Running main ddb53 cgd c3i sparc ..."
echo ""
main ddb53 cgd c3i sparc
```

```
echo -n "Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 cud c3i sparc file2 ..."  
echo "  
main ddb53 cud c3i sparc file2  
echo -n "Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 cgd c3i sparc ..."  
main ddb53 cgd c3i sparc  
echo -n "CONFIGURATION DUMP SUMMARY test complete... Press Return to continue "  
set a=($<)  
clear
```

```
echo "Running main ddb53 cda c3i sparc ..."  
echo "  
main ddb53 cda c3i sparc  
echo "  
echo -n "GET CONFIGURATION DATE test complete... Press Return to continue"  
set a=($<)  
clear
```

```
echo "Running main ddb53 cpl c3i sparc 'This is the first Post to Sparc Log' "  
echo "  
main ddb53 cpl c3i sparc "This is the first Post to Sparc Log"  
echo "  
echo "Now check it to see that it's posted..."  
echo "  
echo "Running ddb53 cgl c3i sparc ..."  
echo "  
main ddb53 cgl c3i sparc  
echo -n "Press Return to continue "  
set a=($<)  
clear
```

```
echo "Add another line to the log ..."  
echo "Running main ddb53 cpl c3i sparc 'This is the second Post to Sparc Log' "  
echo "  
main ddb53 cpl c3i sparc "This is the second Post to Sparc Log"  
echo "  
echo "Show the results ..."  
echo "Running ddb53 cgl c3i sparc ..."  
echo "  
main ddb53 cgl c3i sparc  
echo "  
echo -n "POST/GET CONFIGURATION LOG test complete... Press Return to continue"  
set a=($<)  
clear
```

```
echo "  
echo "Running main ddb53 cgm c3i dos ..."  
main ddb53 cgm c3i dos
```

```

echo "Now we'll change the manager to 'Kelly McDowell' "
echo ""
echo ""
echo "Running main ddb53 cum c3i dos 'Kelly McDowell' ..."
main ddb53 cum c3i dos "Kelly McDowell"
echo ""
echo ""
echo "Now make sure it posted..."
echo "Running main ddb53 cgm c3i dos ..."
main ddb53 cgm c3i dos
echo ""
echo -n "GET/UPDATE CONFIGURATION MANAGER test complete... Press Return to continue"
set a=($<)
clear

```

```

echo "Running main ddb53 cln c3i ..."
main ddb53 cln c3i
echo ""
echo -n "LIST CONFIGURATION NAMES test complete... Press Return to continue"
set a=($<)
clear

```

```

echo "We now move on to the actual loading of the ONTOS Database. "
echo ""
echo "Running main ddb53 vaa c3i c3i ..."
main ddb53 vaa c3i c3i
echo ""
echo "Add c3i test complete... Now we'll add a different Prototype "
echo -n "called fishfarm. Press Return to continue..."
set a=($<)
clear

```

```

setenv PROTOTYPE /n/gemini/work/dwyer/SCCS/fish
echo "Running main ddb53 vaa fishfarm fishfarm ..."
main ddb53 vaa fishfarm fishfarm
echo ""
echo "Add fishfarm test complete... Now we'll add a different Prototype "
echo -n "called robot. Press Return to continue..."
set a=($<)
clear

```

```

setenv PROTOTYPE /n/gemini/work/dwyer/SCCS/robot
echo "Running main ddb53 vaa robot robot ..."
main ddb53 vaa robot robot
echo "Add robot test complete ... Press return to continue"
set a=($<)
clear

```

```

setenv PROTOTYPE /n/gemini/work/dwyer/caps/proto
echo ""
echo "Now show the PLL command to reflect that each Prototype has a default version"
echo ""
echo "Running ddb53 pll ..."
main ddb53 pll
echo -n "LOAD V_OBJECT and SUBTREE test complete... Press Return to continue"
set a=($<)
clear

```

```

echo "Running main ddb53 vlo c3i c3i ..."
echo ""
main ddb53 vlo c3i c3i
echo
echo -n "Press return to continue ..."
set a=($<)
clear

```

```

echo "Running main ddb53 vll c3i c3i ..."
echo ""
main ddb53 vll c3i c3i
echo -n "Press Return to continue ..."
set a=($<)
clear

```

```

echo "Running main ddb53 vlo c3i c3i.comms_interface ..."
echo ""
main ddb53 vlo c3i c3i.comms_interface
echo -n "Press Return to continue ..."
set a=($<)
clear

```

```

echo "Running main ddb53 vll c3i c3i.comms_interface ..."
echo ""
main ddb53 vll c3i c3i.comms_interface
echo ""
echo -n "V_OBJECT LIST/LONG LIST OPERATORS test complete... Press Return to continue"
set a=($<)
clear

```

```

echo "Running main ddb53 vlo c3i c3i.comms_interface.resolve_outgoing_messages ..."
echo ""
main ddb53 vlo c3i c3i.comms_interface.resolve_outgoing_messages
echo -n "Press Return to continue ..."
set a=($<)
clear

```

```

echo "Running main ddb53 vll c3i c3i.comms_interface.resolve_outgoing_messages ..."
echo ""
main ddb53 vll c3i c3i.comms_interface.resolve_outgoing_messages
echo ""
echo -n "V_OBJECT LIST/LONG LIST OPERATORS test complete... Press Return to continue"
set a=($<)
clear

```

```

echo "Running main ddb53 CAO c3i unix c3i ..."
echo ""
main ddb53 cao c3i unix c3i
echo ""
echo "Running main ddb53 cao c3i dos c3i.user_interface ..."
echo ""
main ddb53 cao c3i dos c3i.user_interface
echo "Running main ddb53 cao c3i sparc c3i.comms_interface.resolve_outgoing_messages"
echo ""
main ddb53 cao c3i sparc c3i.comms_interface.resolve_outgoing_messages
echo ""
echo -n "ATTACH OPERATORS to a CONFIGURATION test complete... Press Return to continue"
set a=($<)
clear

```

```
echo "Running main ddb53 clo c3i unix ..."
main ddb53 clo c3i unix
echo ""
echo -n "Press Return to continue"
set a=$(  
clear
```

```
echo "Running main ddb53 clo c3i dos ..."
echo
main ddb53 clo c3i dos
echo ""
echo -n "Press Return to continue"
set a=$(  
clear
```

```
echo "Running main ddb53 clo c3i sparc ..."
echo ""
main ddb53 clo c3i sparc
echo ""
echo -n "CONFIGURATION LIST OPERATOR test complete... Press Return to continue"
set a=$(  
clear
```

```
echo "Running main ddb53 cll c3i unix ..."
echo ""
main ddb53 cll c3i unix
echo ""
echo -n "Press Return to continue"
set a=$(  
clear
```

```
echo "Running main ddb53 cll c3i dos ..."
echo ""
main ddb53 cll c3i dos
echo ""
echo -n "Press Return to continue"
set a=$(  
clear
```

```
echo "Running main ddb53 cll c3i sparc ..."
main ddb53 cll c3i sparc
echo ""
echo -n "CONFIGURATION LONG LIST OPERATORS test complete... Press Return to continue"
set a=$(  
clear
```

```
echo "Running main ddb53 cdt c3i unix r ..."
rm -f /n/gemini/work/dwyer/caps/proto/*
echo ""
main ddb53 cdt c3i unix r
echo ""
echo -n "Examine the directory now and you'll see the checked out subtree ..."
set a=$(  
clear
```

```

echo "Running main ddb53 cdt c3i unix w ...Dumping Tree"
rm -f /n/gemini/work/dwyer/caps/proto/*
echo ""
main ddb53 cdt c3i unix w
echo ""
echo -n "Examine the directory now and you'll see the checked out subtree ..."
set a=($<)
clear

```

```

echo "Running main ddb53 crl c3i unix ...Resetting Locks"
main ddb53 crl c3i unix
echo "Running main ddb53 cdt c3i dos r ...Dumping Tree"
rm -f /n/gemini/work/dwyer/caps/proto/*
echo ""
main ddb53 cdt c3i dos r
echo ""
echo -n "Examine the directory now and you'll see the checked out subtree ..."
set a=($<)
clear

```

```

echo "Running main ddb53 cdt c3i sparc w ...Dumping Tree"
rm -f /n/gemini/work/dwyer/caps/proto/*
echo ""
main ddb53 cdt c3i sparc w
echo ""
echo ""
echo "Examine the directory now and you'll see the checked out subtree ..."
echo -n "Press return to continue "
set a=($<)
clear

```

```

#####
#
#VERSIONING STARTS HERE ...
#
#####

```

```

echo "Running main ddb53 crl c3i unix ...Resetting Locks"
main ddb53 crl c3i sparc
echo "Running main ddb53 cdt c3i unix w ...Dumping Tree for UPDATE"
rm -f /n/gemini/work/dwyer/caps/proto/*
main ddb53 cdt c3i unix w
echo ""
echo ""
echo ""
echo -n "CONFIGURATION DUMP TREE test complete ... Press Return to continue"
set a=($<)
clear

```

```

echo "Simulating updates by TOUCHING files "
echo ""
echo "Running touch c3i.ps ..."
touch /n/gemini/work/dwyer/caps/proto/c3i.ps
echo ""
echo "Running touch c3i.user_interface.imp.psd ..."
echo ""
touch /n/gemini/work/dwyer/caps/proto/c3i.user_interface.imp.psd

```



```

echo ""
echo "Running touch c3i.user_interface.message_arrival_panel.spec.psdl ..."
touch /n/gemini/work/dwyer/caps/proto/c3i.user_interface.message_arrival_panel.spec.psdl
echo ""
echo "Running touch c3i.user_interface.emergency_status_screen.imp.psdl ..."
touch /n/gemini/work/dwyer/caps/proto/c3i.user_interface.emergency_status_screen.imp.psdl
echo ""
echo ""
echo ""
echo "We have just simulated editing several operators...."
echo -n "Press Return to continue"
set a=($<)
clear

echo "Running a CONFIGURATION update now (Posting version #2 of c3i) ..."
echo ""
echo "Running main ddb53 caa c3i unix ..."
echo ""
main ddb53 caa c3i unix
echo ""
echo ""
echo ""
rm -f /n/gemini/work/dwyer/caps/proto/*
echo -n "CONFIGURATION UPDATE TREE test complete ... Press Return to continue"
set a=($<)
clear

echo ""
echo ""
echo ""
echo "Show the versions of the root vobject ..."
echo "Running ddb53 vgv c3i c3i ..."
echo ""
main ddb53 vgv c3i c3i
echo ""
echo ""
echo "Show the versions of the user_interface vobject ..."
echo ""
echo "Running ddb53 vgv c3i c3i.user_interface ..."
echo ""
main ddb53 vgv c3i c3i.user_interface
echo ""
echo "Show the versions of the emergency_status_screen vobject ..."
echo ""
echo "Running ddb53 vgv c3i c3i.user_interface.emergency_status_screen ..."
echo ""
main ddb53 vgv c3i c3i.user_interface.emergency_status_screen
echo ""
echo ""
echo "Show the version of an OPERATOR which DID NOT version ..."
echo ""
echo "Running ddb53 vgv c3i c3i.comms_interface ..."
echo ""
main ddb53 vgv c3i c3i.comms_interface
echo -n "Press Return to continue"
set a=($<)
clear

echo ""
echo ""
echo "Running main ddb53 vdt c3i c3i w "

```

```

echo ""
main ddb53 vdt c3i c3i w

echo "Now updating modules ..."

echo ""
echo "Running touch c3i.ps ..."
touch /n/gemini/work/dwyer/caps/proto/c3i.ps
echo ""
echo "Running touch c3i.user_interface.imp.psdl ..."
echo ""
touch /n/gemini/work/dwyer/caps/proto/c3i.user_interface.imp.psdl
echo ""
echo "Running touch c3i.user_interface.message_arrival_panel.spec.psdl ..."
touch /n/gemini/work/dwyer/caps/proto/c3i.user_interface.message_arrival_panel.spec.psdl
echo ""
echo "Running touch c3i.user_interface.emergency_status_screen.imp.psdl ..."
touch /n/gemini/work/dwyer/caps/proto/c3i.user_interface.emergency_status_screen.imp.psdl
echo ""
echo ""
echo ""
echo "We have just simulated editing several operators...."
echo ""
echo ""
echo "Now Posting Version 3 to the database ..."
echo ""
echo ""
echo "Running main ddb53 vaa c3i c3i "
echo ""
main ddb53 vaa c3i c3i
echo -n "Press Return to continue "
rm -f /n/gemini/work/dwyer/caps/proto/*
set a=($<)
clear

echo ""
echo ""
echo "Now check the versions of c3i in the database "
echo ""
echo ""
echo "Running main ddb53 vgv c3i c3i ..."
echo ""
main ddb53 vgv c3i c3i
echo ""
echo ""
echo "Show the versions of the user_interface vobject ..."
echo ""
echo "Running ddb53 vgv c3i c3i.user_interface ..."
echo ""
main ddb53 vgv c3i c3i.user_interface
echo ""
echo "Show the versions of the emergency_status_screen vobject ..."
echo ""
echo "Running ddb53 vgv c3i c3i.user_interface.emergency_status_screen ..."
echo ""
main ddb53 vgv c3i c3i.user_interface.emergency_status_screen
echo ""
echo ""
echo "Show the version of an OPERATOR which DID NOT version ..."
echo ""
echo "Running ddb53 vgv c3i c3i.comms_interface ..."

```

```

echo ""
main ddb53 vgv c3i c3i.comms_interface
echo -n "Show Versions (VGv) test complete ...Press Return to continue"
set a=($<)
clear

echo "Running main ddb53 vud c3i c3i.user_interface file3 ..."
echo ""
main ddb53 vud c3i c3i.user_interface file3
echo ""
echo ""
echo "Here is the description entered to that operator ..."
echo ""
echo "Running main ddb53 vgd c3i c3i.user_interface ..."
echo ""
main ddb53 vgd c3i c3i.user_interface
echo ""
echo ""
echo -n "Press Return to continue ..."
set a=($<)
clear

echo "Now let's update an OLDER version of an operator previously stored ..."
echo "Running main ddb53 vud c3i c3i.user_interface file4 1 ..."
echo ""
main ddb53 vud c3i c3i.user_interface file4 1
echo ""
echo ""
echo "Here is the description entered to that operator ..."
echo ""
echo "Running main ddb53 vgd c3i c3i.user_interface 1 ..."
echo ""
main ddb53 vgd c3i c3i.user_interface 1
echo ""
echo ""
echo -n "UPDATE VOBJECT/OPERATOR (VUD) test complete... Press Return to continue ..."
set a=($<)
clear

echo "Now attach the configurations to different versions of ROOT VOBJECT ..."
echo "Running main ddb53 cao c3i unix c3i 1 ..."
echo ""
main ddb53 cao c3i unix c3i 1
echo "Running main ddb53 cao c3i dos c3i 2 ..."
echo ""
main ddb53 cao c3i dos c3i 2
echo "Running main ddb53 cao c3i sparc c3i 3 ..."
echo ""
main ddb53 cao c3i sparc c3i 3
echo -n "Press Return to continue ..."
set a=($<)
clear

echo "One Last list to verify everything in order ..."
echo ""
echo "Press Return to continue ..."
set a=($<)
clear

echo "Running main ddb53 clo c3i unix ..."
echo ""

```

```
main ddb53 clo c3i unix
echo -n "Press Return to continue ..."
set a=$(($<))
clear
```

```
echo "Running main ddb53 clo c3i dos"
echo ""
main ddb53 clo c3i dos
echo -n "Press Return to continue ..."
set a=$(($<))
clear
```

```
echo "Running main ddb53 clo c3i sparc"
echo ""
main ddb53 clo c3i sparc
echo -n "Press Return to continue ..."
set a=$(($<))
clear
echo ""
echo ""
echo "*****"
echo " * NAVAL POSTGRADUATE SCHOOL *"
echo " * (THE END) *"
echo " * Software Engineering *"
echo " * Design Database *"
echo " * Thesis Advisor Dr. Luqi"
echo " * By Drew Dwyer and Garry Lewis"
echo " * Design Database v1.1 WP.D.A.L.G *"
echo "*****"
```

C. TEST RESULTS

```
*****  
echo " * NAVAL POSTGRADUATE SCHOOL *"  
echo " * Software Engineering *"  
echo " * Design Database *"  
echo " * Thesis Advisor Dr. Luqi  
echo " * Written by: Drew Dwyer and Garry Lewis *"  
echo " * Design Database v1.1 WP.D.A.L.G . *"  
*****
```

Press Return to begin demonstration

Running main ddb53 pip c3i ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 pip fishfarm 'Drew Dwyer' ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 pip robot 'Garry W. Lewis' file1 ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

INSERT PROTOTYPE test complete ... Press Return to continue

Running main ddb53 pgd robot ...returns description to screen...

```
*****
*****

##### ## # ##### #
# # # #
# # # #
##### # # ##### #
# # # #
# # # #
# # # #
# # # #
```

This is a simple text file
to test Drew and Garry's Prototype Program.

```
*****
*****
```

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 pud robot file2 ...updates description ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 pgd robot ...returns the new description.

```
*****
*****

##### ## # ##### #
# # # #
# # # #
##### # # ##### #
# # # #
# # # #
# # # #
# # # #
```

This file is used to update the
DESCRIPTION ATTRIBUTE
in the Design Database.

<<< FILE UPDATED SUCCESSFULLY >>>>

```
*****
```

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

GET/UPDATE PROTOTYPE DESCRIPTION test complete ... Press Return to continue

Running main ddb53 prd c3i

Tue Sep 3 11:05:42 1991

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

RETRIEVE PROTOTYPE DATE test complete ... Press Return to continue

Running main ddb53 pul robot Larry Williamson ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 pgl robot
Larry Williamson

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

UPDATE/GET PROTOTYPE test complete ... Press Return to continue

Running main ddb53 pds c3i

Creation Date: Tue Sep 3 11:05:42 1991

Leader:

Default Config:

<No configurations are contained in this prototype.>

Prototype Description follows:

=====

<This prototype does not contain a description>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 pds fishfarm ...

Creation Date: Tue Sep 3 11:05:47 1991

Leader: Drew Dwyer
Default Config:

<No configurations are contained in this prototype.>

Prototype Description follows:

=====

<This prototype does not contain a description>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 pds robot ...

Creation Date: Tue Sep 3 11:05:50 1991

Leader: Larry Williamson
Default Config:

<No configurations are contained in this prototype.>

Prototype Description follows:

=====

#####

This file is used to update the
DESCRIPTION ATTRIBUTE
in the Design Database.

<<< FILE UPDATED SUCCESSFULLY >>>>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

DUMP PROTOTYPE SUMMARY test complete ... Press Return to continue

Running main ddb53 pln ...

c3i
fishfarm
robot

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

LIST PROTOTYPE NAMES test complete ... Press Return to continue

Running main ddb53 cic c3i unix ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 cic c3i dos 'Drew Dwyer' ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 cic c3i sparc 'Garry Lewis' file1 ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

INSERT CONFIGURATION test complete... Press Return to continue

Running main ddb53 cum c3i dos 'Marty Shoppenheimer' ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

UPDATE CONFIGURATION MANAGER test complete... Press Return to continue

Running main ddb53 cds c3i unix ...
Creation Date: Tue Sep 3 11:08:26 1991

Manager:
Version Number: NONE

Default VOBJECT Name

=====

<This configuration has not been assigned a V_OBJECT>
Configuration Description follows:

=====

<This configuration does not contain a description>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 cds c3i dos ...
Creation Date: Tue Sep 3 11:08:28 1991

Manager: Marty Shoppenheimer
Version Number: NONE

Default VOBJECT Name

<This configuration has not been assigned a V_OBJECT>
Configuration Description follows:

<This configuration does not contain a description>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 cds c3i sparc ...
Creation Date: Tue Sep 3 11:08:31 1991

Manager: Garry Lewis
Version Number: NONE

Default VOBJECT Name

<This configuration has not been assigned a V_OBJECT>
Configuration Description follows:

#####

This is a simple text file
to test Drew and Garry's Prototype Program.

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

DUMP CONFIGURATION SUMMARY test complete... Press Return to continue

Running main ddb53 cgd c3i sparc ...

```
*****
*****
```

```
##### ## # ##### #
# # # #
# # # #
##### # # ##### #
# # # #
# # # #
# # # #
# # # #
```

This is a simple text file

to test Drew and Garry's Prototype Program.

```
*****
*****
```

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 cud c3i sparc file2 ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 cgd c3i sparc ...

```
*****
*****
```

```
##### ## # ##### #####
# # # #
# # # #
##### # # ##### #####
# # # #
# # # #
# # # #
# # # #
```

This file is used to update the
DESCRIPTION ATTRIBUTE
in the Design Database.

<<< FILE UPDATED SUCCESSFULLY >>>>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

CONFIGURATION DUMP SUMMARY test complete... Press Return to continue

Running main ddb53 cda c3i sparc ...

Tue Sep 3 11:08:31 1991

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

GET CONFIGURATION DATE test complete... Press Return to continue

Running main ddb53 cpl c3i sparc 'This is the first Post to Sparc Log'

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Now check it to see that it's posted...

Running ddb53 cgl c3i sparc ...

Tue Sep 3 11:09:50 1991

This is the first Post to Sparc Log

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Add another line to the log ...

Running main ddb53 cpl c3i sparc 'This is the second Post to Sparc Log'

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show the results ...

Running ddb53 cgl c3i sparc ...

Tue Sep 3 11:09:50 1991

This is the first Post to Sparc Log

Tue Sep 3 11:09:56 1991

This is the second Post to Sparc Log

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

POST/GET CONFIGURATION LOG test complete... Press Return to continue

Running main ddb53 cgm c3i dos ...
Marty Shoppenheimer

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Now we'll change the manager to 'Kelly Mcdowell'

Running main ddb53 cum c3i dos 'Kelly Mcdowell' ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Now make sure it posted...
Running main ddb53 cgm c3i dos ...
Kelly Mcdowell

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

GET/UPDATE CONFIGURATION MANAGER test complete... Press Return to continue

Running main ddb53 cln c3i ...
unix
dos
sparc

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

LIST CONFIGURATION NAMES test complete... Press Return to continue

We now move on to the actual loading of the ONTOS Database.

Running main ddb53 vaa c3i c3i ...
CHECKIN--> c3i
CHECKIN--> c3i.comms_interface
CHECKIN--> c3i.comms_interface.prepare_periodic_report

CHECKIN--> c3i.comms_interface.resolve_incoming_messages
 CHECKIN--> c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
 CHECKIN--> c3i.comms_interface.resolve_incoming_messages.decide_for_relaying
 CHECKIN--> c3i.comms_interface.resolve_incoming_messages.extract_tracks
 CHECKIN--> c3i.comms_interface.resolve_incoming_messages.parse_input_file
 CHECKIN--> c3i.comms_interface.resolve_outgoing_messages
 CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
 CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
 CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
 CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.make_routing
 CHECKIN--> c3i.comms_interface.translate_message
 CHECKIN--> c3i.comms_links
 CHECKIN--> c3i.navigation_system
 CHECKIN--> c3i.sensor_interface
 CHECKIN--> c3i.sensor_interface.analyze_sensor_data
 CHECKIN--> c3i.sensor_interface.normalize_sensor_information
 CHECKIN--> c3i.sensor_interface.prepare_sensor_track
 CHECKIN--> c3i.sensors
 CHECKIN--> c3i.track_database_manager
 CHECKIN--> c3i.track_database_manager.monitor_database
 CHECKIN--> c3i.track_database_manager.update_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.add_comms_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.add_sensor_track
 CHECKIN--> c3i.track_database_manager.update_tracks.add_user_track
 CHECKIN--> c3i.track_database_manager.update_tracks.delete_the_track
 CHECKIN--> c3i.track_database_manager.update_tracks.filter_comms_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.filter_sensor_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.monitor_ownership_position
 CHECKIN--> c3i.track_database_manager.update_tracks.update_the_track
 CHECKIN--> c3i.user_interface
 CHECKIN--> c3i.user_interface.display_graphic_tracks
 CHECKIN--> c3i.user_interface.display_tracks
 CHECKIN--> c3i.user_interface.emergency_status_screen
 CHECKIN--> c3i.user_interface.get_modification_data
 CHECKIN--> c3i.user_interface.get_user_inputs
 CHECKIN--> c3i.user_interface.intelligence_report_panel
 CHECKIN--> c3i.user_interface.manage_user_interface
 CHECKIN--> c3i.user_interface.message_arrival
 CHECKIN--> c3i.user_interface.message_arrival_panel
 CHECKIN--> c3i.user_interface.message_editor
 CHECKIN--> c3i.user_interface.resolution_notice_panel
 CHECKIN--> c3i.user_interface.status_screen
 CHECKIN--> c3i.weapons_interface
 CHECKIN--> c3i.weapons_systems

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Add c3i test complete... Now we'll add a different Prototype called fishfarm. Press Return to continue...

Running main ddb53 vaa fishfarm fishfarm ...

CHECKIN--> fishfarm
 CHECKIN--> fishfarm.Control_feeding
 CHECKIN--> fishfarm.Control_feeding_times
 CHECKIN--> fishfarm.Determine_actual_inlet_valve_setting
 CHECKIN--> fishfarm.Determine_actual_outlet_valve_setting
 CHECKIN--> fishfarm.H2O_sensor
 CHECKIN--> fishfarm.Determine_desired_inlet_valve_setting_NH3

```
CHECKIN--> fishfarm.Determine_desired_inlet_valve_setting_O2
CHECKIN--> fishfarm.Display_system_status
CHECKIN--> fishfarm.Inlet_valve
CHECKIN--> fishfarm.NH3_sensor
CHECKIN--> fishfarm.Outlet_valve
CHECKIN--> fishfarm.ffconsole
CHECKIN--> fishfarm.mytime
CHECKIN--> fishfarm.o2_sensor
```

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Add fishfarm test complete... Now we'll add a different Prototype called robot. Press Return to continue...

```
Running main ddb53 vaa robot robot ...
CHECKIN--> robot
CHECKIN--> robot.Accelerometer
CHECKIN--> robot.Calculate_Position
CHECKIN--> robot.Fire_Thrusters
CHECKIN--> robot.Get_Keys
CHECKIN--> robot.Update_Acceleration
CHECKIN--> robot.Update_Display
CHECKIN--> robot.Update_Thrust_Req
```

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Add robot test complete ... Press return to continue

Now show the PLL command to reflect that each Prototype has a default version

```
Running ddb53 pll ...
Name: c3i Default Config: sparc Version: 1
Name: fishfarm Default Config: NONE Version: 1
Name: robot Default Config: NONE Version: 1
```

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

LOAD V_OBJECT and SUBTREE test complete... Press Return to continue

Running main ddb53 vlo c3i c3i ...

```
Operator: c3i
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969
```

```
Operator: c3i.comms_interface.prepare_periodic_report
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969
```

```
Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969
```

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_relaying
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.extract_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.parse_input_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.translate_message
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_links
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.navigation_system
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.analyze_sensor_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.normalize_sensor_information
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.prepare_sensor_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensors
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.monitor_database
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_comms_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_sensor_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_user_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.delete_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_comms_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_sensor_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.monitor_ownership_position
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.update_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_graphic_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_tracks

Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.emergency_status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_modification_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_user_inputs
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.intelligence_report_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.manage_user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_editor
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.resolution_notice_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_systems
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press return to continue ...

Running main ddb53 vll c3i c3i ...

Operator: comms_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: comms_links
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: navigation_system
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: sensor_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: sensors
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: track_database_manager
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: weapons_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: weapons_systems
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 199;(c) WP.D.A.L.G

Press Return to continue ...

Running main ddb53 vlo c3i c3i.comms_interface .

Operator: c3i.comms_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.prepare_periodic_report
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_relaying

Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.extract_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.parse_input_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.translate_message
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue ...

Running main ddb53 vll c3i c3i.comms_interface ...

Operator: prepare_periodic_report
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: resolve_incoming_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: translate_message

Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

V_OBJECT LIST/LONG LIST OPERATORS test complete... Press Return to continue

Running main ddb53 vlo c3i c3i.comms_interface.resolve_outgoing_messages ...

Operator: c3i.comms_interface.resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue ...

Running main ddb53 vll c3i c3i.comms_interface.resolve_outgoing_messages ...

Operator: convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

V_OBJECT LIST/LONG LIST OPERATORS test complete... Press Return to continue

Running main ddb53 CAO c3i unix c3i ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 cao c3i dos c3i.user_interface

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 cao c3i sparc c3i.comms_interface.resolve_outgoing_messages

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

ATTACH OPERATORS to a CONFIGURATION test complete... Press Return to continue

Running main ddb53 clo c3i unix ...

Operator: c3i

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.prepare_periodic_report

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_archiving

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_relaying

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.extract_tracks

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.parse_input_file

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.translate_message

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_links

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.navigation_system

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.analyze_sensor_data

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.normalize_sensor_information

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.prepare_sensor_track

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensors

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.monitor_database

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_comms_tracks

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_sensor_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_user_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.delete_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_comms_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_sensor_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.monitor_ownership_position
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.update_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_graphic_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.emergency_status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_modification_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_user_inputs
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.intelligence_report_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.manage_user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_editor
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.resolution_notice_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_systems
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 clo c3i dos ...
Operator: c3i.user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_graphic_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.emergency_status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_modification_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_user_inputs
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.intelligence_report_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.manage_user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_editor
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.resolution_notice_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 clo c3i sparc ...

Operator: c3i.comms_interface.resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

CONFIGURATION LIST OPERATOR test complete... Press Return to continue

Running main ddb53 cll c3i unix ...

Operator: comms_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: comms_links
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: navigation_system
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: sensor_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: sensors
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: track_database_manager
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: weapons_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: weapons_systems
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 cll c3i dos ...

Operator: display_graphic_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: display_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: emergency_status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: get_modification_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: get_user_inputs
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: intelligence_report_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: manage_user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: message_arrival
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: message_arrival_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: message_editor
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: resolution_notice_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Running main ddb53 cli c3i sparc ...
Operator: convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

CONFIGURATION LONG LIST OPERATORS test complete... Press Return to continue

Running main ddb53 cdt c3i unix r ...

NODENAME ---> c3i
Version: 1

NODENAME ---> prepare_periodic_report
Version: 1

NODENAME ---> decide_for_archiving
Version: 1

NODENAME ---> decide_for_relaying
Version: 1

NODENAME ---> extract_tracks
Version: 1

NODENAME ---> parse_input_file
Version: 1

NODENAME ---> resolve_incoming_messages
Version: 1

NODENAME ---> convert_to_text_file
Version: 1

NODENAME ---> forward_for_translation
Version: 1

NODENAME ---> forward_for_transmission
Version: 1

NODENAME ---> make_routing
Version: 1

NODENAME ---> resolve_outgoing_messages
Version: 1

NODENAME ---> translate_message
Version: 1

NODENAME ---> comms_interface
Version: 1

NODENAME ---> comms_links
Version: 1

NODENAME ---> navigation_system
Version: 1

NODENAME ---> analyze_sensor_data
Version: 1

NODENAME ---> normalize_sensor_information
Version: 1

NODENAME ---> prepare_sensor_track
Version: 1

NODENAME ---> sensor_interface
Version: 1

NODENAME ---> sensors
Version: 1

NODENAME ---> monitor_database
Version: 1

NODENAME ---> add_comms_tracks
Version: 1

NODENAME ---> add_sensor_track
Version: 1

NODENAME ---> add_user_track
Version: 1

NODENAME ---> delete_the_track
Version: 1

NODENAME ---> filter_comms_tracks
Version: 1

NODENAME ---> filter_sensor_tracks
Version: 1

NODENAME ---> monitor_ownership_position
Version: 1

NODENAME ---> update_the_track
Version: 1

NODENAME ---> update_tracks
Version: 1

NODENAME ---> track_database_manager
Version: 1

NODENAME ---> display_graphic_tracks
Version: 1

NODENAME ---> display_tracks

Version: 1

NODENAME ---> emergency_status_screen
Version: 1

NODENAME ---> get_modification_data
Version: 1

NODENAME ---> get_user_inputs
Version: 1

NODENAME ---> intelligence_report_panel
Version: 1

NODENAME ---> manage_user_interface
Version: 1

NODENAME ---> message_arrival
Version: 1

NODENAME ---> message_arrival_panel
Version: 1

NODENAME ---> message_editor
Version: 1

NODENAME ---> resolution_notice_panel
Version: 1

NODENAME ---> status_screen
Version: 1

NODENAME ---> user_interface
Version: 1

NODENAME ---> weapons_interface
Version: 1

NODENAME ---> weapons_systems
Version: 1

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Examine the directory now and you'll see the checked out subtree ...

Running main ddb53 cdt c3i unix w ...Dumping Tree

NODENAME ---> c3i
Version: 1

setworker --> to dwyer
NODENAME ---> prepare_periodic_report
Version: 1

setworker --> to dwyer
NODENAME ---> decide_for_archiving
Version: 1

setworker --> to dwyer
NODENAME ---> decide_for_relaying
Version: 1

setworker --> to dwyer
NODENAME ---> extract_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> parse_input_file
Version: 1

setworker --> to dwyer
NODENAME ---> resolve_incoming_messages
Version: 1

setworker --> to dwyer
NODENAME ---> convert_to_text_file
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_translation
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_transmission
Version: 1

setworker --> to dwyer
NODENAME ---> make_routing
Version: 1

setworker --> to dwyer
NODENAME ---> resolve_outgoing_messages
Version: 1

setworker --> to dwyer
NODENAME ---> translate_message
Version: 1

setworker --> to dwyer
NODENAME ---> comms_interface
Version: 1

setworker --> to dwyer
NODENAME ---> comms_links
Version: 1

setworker --> to dwyer
NODENAME ---> navigation_system
Version: 1

setworker --> to dwyer
NODENAME ---> analyze_sensor_data
Version: 1

setworker --> to dwyer
NODENAME ---> normalize_sensor_information
Version: 1

setworker --> to dwyer
NODENAME ---> prepare_sensor_track
Version: 1

setworker --> to dwyer
NODENAME ---> sensor_interface
Version: 1

setworker --> to dwyer
NODENAME ---> sensors
Version: 1

setworker --> to dwyer
NODENAME ---> monitor_database
Version: 1

setworker --> to dwyer
NODENAME ---> add_comms_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> add_sensor_track
Version: 1

setworker --> to dwyer
NODENAME ---> add_user_track
Version: 1

setworker --> to dwyer
NODENAME ---> delete_the_track
Version: 1

setworker --> to dwyer
NODENAME ---> filter_comms_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> filter_sensor_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> monitor_ownership_position
Version: 1

setworker --> to dwyer
NODENAME ---> update_the_track
Version: 1

setworker --> to dwyer
NODENAME ---> update_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> track_database_manager
Version: 1

setworker --> to dwyer
NODENAME ---> display_graphic_tracks
Version: 1

setworker --> to dwyer

NODENAME ---> display_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> emergency_status_screen
Version: 1

setworker --> to dwyer
NODENAME ---> get_modification_data
Version: 1

setworker --> to dwyer
NODENAME ---> get_user_inputs
Version: 1

setworker --> to dwyer
NODENAME ---> intelligence_report_panel
Version: 1

setworker --> to dwyer
NODENAME ---> manage_user_interface
Version: 1

setworker --> to dwyer
NODENAME ---> message_arrival
Version: 1

setworker --> to dwyer
NODENAME ---> message_arrival_panel
Version: 1

setworker --> to dwyer
NODENAME ---> message_editor
Version: 1

setworker --> to dwyer
NODENAME ---> resolution_notice_panel
Version: 1

setworker --> to dwyer
NODENAME ---> status_screen
Version: 1

setworker --> to dwyer
NODENAME ---> user_interface
Version: 1

setworker --> to dwyer
NODENAME ---> weapons_interface
Version: 1

setworker --> to dwyer
NODENAME ---> weapons_systems
Version: 1

setworker --> to dwyer

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 crl c3i unix ...Resetting Locks

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Examine the directory now and you'll see the checked out subtree ...

Running main ddb53 cdt c3i dos r ...Dumping Tree

NODENAME ---> user_interface
Version: 1

NODENAME ---> display_graphic_tracks
Version: 1

NODENAME ---> display_tracks
Version: 1

NODENAME ---> emergency_status_screen
Version: 1

NODENAME ---> get_modification_data
Version: 1

NODENAME ---> get_user_inputs
Version: 1

NODENAME ---> intelligence_report_panel
Version: 1

NODENAME ---> manage_user_interface
Version: 1

NODENAME ---> message_arrival
Version: 1

NODENAME ---> message_arrival_panel
Version: 1

NODENAME ---> message_editor
Version: 1

NODENAME ---> resolution_notice_panel
Version: 1

NODENAME ---> status_screen
Version: 1

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Examine the directory now and you'll see the checked out subtree ...

Running main ddb53 cdt c3i sparc w ...Dumping Tree

NODENAME ---> resolve_outgoing_messages

Version: 1

setworker --> to dwyer
NODENAME ---> convert_to_text_file
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_translation
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_transmission
Version: 1

setworker --> to dwyer
NODENAME ---> make_routing
Version: 1

setworker --> to dwyer

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 crl c3i unix ...Resetting Locks

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Examine the directory now and you'll see the checked out subtree ...
Press return to continue

Running main ddb53 cdt c3i unix w ...Dumping Tree for UPDATE
NODENAME ---> c3i
Version: 1

setworker --> to dwyer
NODENAME ---> prepare_periodic_report
Version: 1

setworker --> to dwyer
NODENAME ---> decide_for_archiving
Version: 1

setworker --> to dwyer
NODENAME ---> decide_for_relaying
Version: 1

setworker --> to dwyer
NODENAME ---> extract_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> parse_input_file
Version: 1

setworker --> to dwyer
NODENAME ---> resolve_incoming_messages

Version: 1

setworker --> to dwyer
NODENAME ---> convert_to_text_file
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_translation
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_transmission
Version: 1

setworker --> to dwyer
NODENAME ---> make_routing
Version: 1

setworker --> to dwyer
NODENAME ---> resolve_outgoing_messages
Version: 1

setworker --> to dwyer
NODENAME ---> translate_message
Version: 1

setworker --> to dwyer
NODENAME ---> comms_interface
Version: 1

setworker --> to dwyer
NODENAME ---> comms_links
Version: 1

setworker --> to dwyer
NODENAME ---> navigation_system
Version: 1

setworker --> to dwyer
NODENAME ---> analyze_sensor_data
Version: 1

setworker --> to dwyer
NODENAME ---> normalize_sensor_information
Version: 1

setworker --> to dwyer
NODENAME ---> prepare_sensor_track
Version: 1

setworker --> to dwyer
NODENAME ---> sensor_interface
Version: 1

setworker --> to dwyer
NODENAME ---> sensors
Version: 1

setworker --> to dwyer
NODENAME ---> monitor_database
Version: 1

setworker --> to dwyer
NODENAME ---> add_comms_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> add_sensor_track
Version: 1

setworker --> to dwyer
NODENAME ---> add_user_track
Version: 1

setworker --> to dwyer
NODENAME ---> delete_the_track
Version: 1

setworker --> to dwyer
NODENAME ---> filter_comms_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> filter_sensor_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> monitor_ownership_position
Version: 1

setworker --> to dwyer
NODENAME ---> update_the_track
Version: 1

setworker --> to dwyer
NODENAME ---> update_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> track_database_manager
Version: 1

setworker --> to dwyer
NODENAME ---> display_graphic_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> display_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> emergency_status_screen
Version: 1

setworker --> to dwyer
NODENAME ---> get_modification_data
Version: 1

setworker --> to dwyer
NODENAME ---> get_user_inputs
Version: 1

setworker --> to dwyer
NODENAME ---> intelligence_report_panel
Version: 1

setworker --> to dwyer
NODENAME ---> manage_user_interface
Version: 1

setworker --> to dwyer
NODENAME ---> message_arrival
Version: 1

setworker --> to dwyer
NODENAME ---> message_arrival_panel
Version: 1

setworker --> to dwyer
NODENAME ---> message_editor
Version: 1

setworker --> to dwyer
NODENAME ---> resolution_notice_panel
Version: 1

setworker --> to dwyer
NODENAME ---> status_screen
Version: 1

setworker --> to dwyer
NODENAME ---> user_interface
Version: 1

setworker --> to dwyer
NODENAME ---> weapons_interface
Version: 1

setworker --> to dwyer
NODENAME ---> weapons_systems
Version: 1

setworker --> to dwyer

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

CONFIGURATION DUMP TREE test complete ... Press Return to continue

Simulating updates by TOUCHING files

Running touch c3i.ps ...

Running touch c3i.user_interface.imp.psd1 ...

Running touch c3i.user_interface.message_arrival_panel.spec.psd1 ...

Running touch c3i.user_interface_emergency_status_screen.imp.psd1 ...

We have just simulated editing several operators....
Press Return to continue

Running a CONFIGURATION update now (Posting version #2 of c3i) ...

Running main ddb53 caa c3i unix ...

```
CHECKIN--> c3i
CHECKIN--> c3i.comms_interface
CHECKIN--> c3i.comms_interface.prepare_periodic_report
CHECKIN--> c3i.comms_interface.resolve_incoming_messages
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.decide_for_relaying
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.extract_tracks
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.parse_input_file
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.make_routing
CHECKIN--> c3i.comms_interface.translate_message
CHECKIN--> c3i.comms_links
CHECKIN--> c3i.navigation_system
CHECKIN--> c3i.sensor_interface
CHECKIN--> c3i.sensor_interface.analyze_sensor_data
CHECKIN--> c3i.sensor_interface.normalize_sensor_information
CHECKIN--> c3i.sensor_interface.prepare_sensor_track
CHECKIN--> c3i.sensors
CHECKIN--> c3i.track_database_manager
CHECKIN--> c3i.track_database_manager.monitor_database
CHECKIN--> c3i.track_database_manager.update_tracks
CHECKIN--> c3i.track_database_manager.update_tracks.add_comms_tracks
CHECKIN--> c3i.track_database_manager.update_tracks.add_sensor_track
CHECKIN--> c3i.track_database_manager.update_tracks.add_user_track
CHECKIN--> c3i.track_database_manager.update_tracks.delete_the_track
CHECKIN--> c3i.track_database_manager.update_tracks.filter_comms_tracks
CHECKIN--> c3i.track_database_manager.update_tracks.filter_sensor_tracks
CHECKIN--> c3i.track_database_manager.update_tracks.monitor_ownership_position
CHECKIN--> c3i.track_database_manager.update_tracks.update_the_track
CHECKIN--> c3i.user_interface
CHECKIN--> c3i.user_interface.display_graphic_tracks
CHECKIN--> c3i.user_interface.display_tracks
CHECKIN--> c3i.user_interface.emergency_status_screen
CHECKIN--> c3i.user_interface.get_modification_data
CHECKIN--> c3i.user_interface.get_user_inputs
CHECKIN--> c3i.user_interface.intelligence_report_panel
CHECKIN--> c3i.user_interface.manage_user_interface
CHECKIN--> c3i.user_interface.message_arrival
CHECKIN--> c3i.user_interface.message_arrival_panel
CHECKIN--> c3i.user_interface.message_editor
CHECKIN--> c3i.user_interface.resolution_notice_panel
CHECKIN--> c3i.user_interface.status_screen
CHECKIN--> c3i.weapons_interface
CHECKIN--> c3i.weapons_systems
```

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

CONFIGURATION UPDATE TREE test complete ... Press Return to continue

Show the versions of the root vobject ...
Running ddb53 vgv c3i c3i ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1
2

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show the versions of the user_interface vobject ...

Running ddb53 vgv c3i c3i.user_interface ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1
2

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show the versions of the emergency_status_screen vobject ...

Running ddb53 vgv c3i c3i.user_interface.emergency_status_screen ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1
2

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show the version of an OPERATOR which DID NOT version ...

Running ddb53 vgv c3i c3i.comms_interface ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1

Press Return to continue

Running main ddb53 vdt c3i c3i w

NODENAME ---> c3i
Version: 2

setworker --> to dwyer
NODENAME ---> prepare_periodic_report
Version: 1

setworker --> to dwyer
NODENAME ---> decide_for_archiving
Version: 1

setworker --> to dwyer
NODENAME ---> decide_for_relaying
Version: 1

setworker --> to dwyer
NODENAME ---> extract_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> parse_input_file
Version: 1

setworker --> to dwyer
NODENAME ---> resolve_incoming_messages
Version: 1

setworker --> to dwyer
NODENAME ---> convert_to_text_file
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_translation
Version: 1

setworker --> to dwyer
NODENAME ---> forward_for_transmission
Version: 1

setworker --> to dwyer
NODENAME ---> make_routing
Version: 1

setworker --> to dwyer
NODENAME ---> resolve_outgoing_messages
Version: 1

setworker --> to dwyer
NODENAME ---> translate_message
Version: 1

setworker --> to dwyer

NODENAME ---> comms_interface
Version: 1

setworker --> to dwyer
NODENAME ---> comms_links
Version: 1

setworker --> to dwyer
NODENAME ---> navigation_system
Version: 1

setworker --> to dwyer
NODENAME ---> analyze_sensor_data
Version: 1

setworker --> to dwyer
NODENAME ---> normalize_sensor_information
Version: 1

setworker --> to dwyer
NODENAME ---> prepare_sensor_track
Version: 1

setworker --> to dwyer
NODENAME ---> sensor_interface
Version: 1

setworker --> to dwyer
NODENAME ---> sensors
Version: 1

setworker --> to dwyer
NODENAME ---> monitor_database
Version: 1

setworker --> to dwyer
NODENAME ---> add_comms_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> add_sensor_track
Version: 1

setworker --> to dwyer
NODENAME ---> add_user_track
Version: 1

setworker --> to dwyer
NODENAME ---> delete_the_track
Version: 1

setworker --> to dwyer
NODENAME ---> filter_comms_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> filter_sensor_tracks
Version: 1

setworker --> to dwyer

NODENAME ---> monitor_ownership_position
Version: 1

setworker --> to dwyer
NODENAME ---> update_the_track
Version: 1

setworker --> to dwyer
NODENAME ---> update_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> track_database_manager
Version: 1

setworker --> to dwyer
NODENAME ---> display_graphic_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> display_tracks
Version: 1

setworker --> to dwyer
NODENAME ---> emergency_status_screen
Version: 2

setworker --> to dwyer
NODENAME ---> get_modification_data
Version: 1

setworker --> to dwyer
NODENAME ---> get_user_inputs
Version: 1

setworker --> to dwyer
NODENAME ---> intelligence_report_panel
Version: 1

setworker --> to dwyer
NODENAME ---> manage_user_interface
Version: 1

setworker --> to dwyer
NODENAME ---> message_arrival
Version: 1

setworker --> to dwyer
NODENAME ---> message_arrival_panel
Version: 2

setworker --> to dwyer
NODENAME ---> message_editor
Version: 1

setworker --> to dwyer
NODENAME ---> resolution_notice_panel
Version: 1

setworker --> to dwyer
NODENAME ---> status_screen

Version: 1

setworker --> to dwyer
NODENAME ---> user_interface
Version: 2

setworker --> to dwyer
NODENAME ---> weapons_interface
Version: 1

setworker --> to dwyer
NODENAME ---> weapons_systems
Version: 1

setworker --> to dwyer

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Now updating modules ...

Running touch c3i.ps ...

Running touch c3i.user_interface.imp.psdl ...

Running touch c3i.user_interface.message_arrival_panel.spec.psdl ...

Running touch c3i.user_interface_emergency_status_screen.imp.psdl ...

We have just simulated editing several operators....

Now Posting Version 3 to the database ...

Running main ddb53 vaa c3i c3i

CHECKIN--> c3i
CHECKIN--> c3i.comms_interface
CHECKIN--> c3i.comms_interface.prepare_periodic_report
CHECKIN--> c3i.comms_interface.resolve_incoming_messages
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.decide_for_relaying
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.extract_tracks
CHECKIN--> c3i.comms_interface.resolve_incoming_messages.parse_input_file
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
CHECKIN--> c3i.comms_interface.resolve_outgoing_messages.make_routing
CHECKIN--> c3i.comms_interface.translate_message
CHECKIN--> c3i.comms_links
CHECKIN--> c3i.navigation_system
CHECKIN--> c3i.sensor_interface
CHECKIN--> c3i.sensor_interface.analyze_sensor_data
CHECKIN--> c3i.sensor_interface.normalize_sensor_information
CHECKIN--> c3i.sensor_interface.prepare_sensor_track

CHECKIN--> c3i.sensors
 CHECKIN--> c3i.track_database_manager
 CHECKIN--> c3i.track_database_manager.monitor_database
 CHECKIN--> c3i.track_database_manager.update_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.add_comms_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.add_sensor_track
 CHECKIN--> c3i.track_database_manager.update_tracks.add_user_track
 CHECKIN--> c3i.track_database_manager.update_tracks.delete_the_track
 CHECKIN--> c3i.track_database_manager.update_tracks.filter_comms_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.filter_sensor_tracks
 CHECKIN--> c3i.track_database_manager.update_tracks.monitor_ownership_position
 CHECKIN--> c3i.track_database_manager.update_tracks.update_the_track
 CHECKIN--> c3i.user_interface
 CHECKIN--> c3i.user_interface.display_graphic_tracks
 CHECKIN--> c3i.user_interface.display_tracks
 CHECKIN--> c3i.user_interface.emergency_status_screen
 CHECKIN--> c3i.user_interface.get_modification_data
 CHECKIN--> c3i.user_interface.get_user_inputs
 CHECKIN--> c3i.user_interface.intelligence_report_panel
 CHECKIN--> c3i.user_interface.manage_user_interface
 CHECKIN--> c3i.user_interface.message_arrival
 CHECKIN--> c3i.user_interface.message_arrival_panel
 CHECKIN--> c3i.user_interface.message_editor
 CHECKIN--> c3i.user_interface.resolution_notice_panel
 CHECKIN--> c3i.user_interface.status_screen
 CHECKIN--> c3i.weapons_interface
 CHECKIN--> c3i.weapons_systems

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

Now check the versions of c3i in the database

Running main ddb53 vgv c3i c3i ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1
 2
 3

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show the versions of the user_interface vobject ...

Running ddb53 vgv c3i c3i.user_interface ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1
 2

3

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show the versions of the emergency_status_screen vobject ...

Running ddb53 vgv c3i c3i.user_interface.emergency_status_screen ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1
2
3

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show the version of an OPERATOR which DID NOT version ...

Running ddb53 vgv c3i c3i.comms_interface ...

THIS THREAD CONTAINS THE FOLLOWING VERSIONS:

1

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Show Versions (VGV) test complete ...Press Return to continue

Running main ddb53 vud c3i c3i.user_interface file3 ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Here is the description entered to that operator ...

Running main ddb53 vgd c3i c3i.user_interface ...

#####

#####

#####

This file is a test file used
to update the VOBJECT
DESCRIPTION ATTRIBUTE.

<<<< UPDATED SUCCESSFULLY >>>>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue ...

Now let's update an OLDER version of an operator previously stored ...
Running main ddb53 vud c3i c3i.user_interface file4 1 ...

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Here is the description entered to that operator ...

Running main ddb53 vgd c3i c3i.user_interface 1 ...

#####

#####

#####

#####

#####

This file is a test file used
to update the VOBJECT
DESCRIPTION ATTRIBUTE.

<<<< UPDATED SUCCESSFULLY >>>>

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

UPDATE VOBJECT/OPERATOR (VUD) test complete... Press Return to continue ...

Now attach the configurations to different versions of ROOT VOBJECT ...
Running main ddb53 cao c3i unix c3i 1

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 cao c3i dos c3i 2

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Running main ddb53 cao c3i sparc c3i 3

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue

One Last list to verify everythings in order ...

Press Return to continue ...

Running main ddb53 clo c3i unix

Operator: c3i
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.prepare_periodic_report

Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_relaying
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.extract_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.parse_input_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.translate_message
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_links
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.navigation_system
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.analyze_sensor_data
Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.normalize_sensor_information

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.prepare_sensor_track

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensors

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.monitor_database

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_comms_tracks

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_sensor_track

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_user_track

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.delete_the_track

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_comms_tracks

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_sensor_tracks

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.monitor_ownership_position

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.update_the_track

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager

Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_graphic_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.emergency_status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_modification_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_user_inputs
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.intelligence_report_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.manage_user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_editor
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.resolution_notice_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_systems
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue ...

Running main ddb53 clo c3i dos

Operator: c3i
Version: 2
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.prepare_periodic_report
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_relaying
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.extract_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.parse_input_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.translate_message
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_links
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.navigation_system
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.analyze_sensor_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.normalize_sensor_information
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.prepare_sensor_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensors
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.monitor_database
Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_comms_tracks
Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_sensor_track
Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_user_track
Version: 1

Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.delete_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_comms_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_sensor_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.monitor_ownership_position
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.update_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_graphic_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.emergency_status_screen
Version: 2
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_modification_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_user_inputs
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.intelligence_report_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.manage_user_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival_panel
Version: 2
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_editor
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.resolution_notice_panel
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.status_screen
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface
Version: 2
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_systems
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue ...

Running main ddb53 clo c3i sparc

Operator: c3i
Version: 3
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.prepare_periodic_report
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_archiving
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.decide_for_relaying
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.extract_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages.parse_input_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_incoming_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.convert_to_text_file
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_translation
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.forward_for_transmission
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages.make_routing
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.resolve_outgoing_messages
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface.translate_message
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.comms_links
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.navigation_system
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.analyze_sensor_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.normalize_sensor_information
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface.prepare_sensor_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensor_interface
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.sensors
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.monitor_database
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_comms_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_sensor_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.add_user_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.delete_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_comms_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.filter_sensor_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.monitor_ownership_position
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks.update_the_track
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager.update_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.track_database_manager
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_graphic_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.display_tracks
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.emergency_status_screen
Version: 3
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_modification_data
Version: 1
Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.get_user_inputs

Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.intelligence_report_panel
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.manage_user_interface
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_arrival_panel
 Version: 3
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.message_editor
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.resolution_notice_panel
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface.status_screen
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.user_interface
 Version: 3
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_interface
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

Operator: c3i.weapons_systems
 Version: 1
 Locktime is: Wed Dec 31 16:00:00 1969

CAPS Engineering Design Database 1991(c) WP.D.A.L.G

Press Return to continue ...

```
*****
echo "* NAVAL POSTGRADUATE SCHOOL *"
echo "* Software Engineering *"
echo "* Design Database *"
echo "* Thesis Advisor Dr. Luqi
echo "* Written by: Drew Dwyer and Garry Lewis *"
echo "* Design Database v1.1 WP.D.A.L.G . *"
*****
[7msun53:/n/gemini/work/dwyer/SCCS>>[m exit
exit
```

script done on Tue Sep 3 11:36:13 1991

APPENDIX D

CODE

A. PREPARATION NOTES

This code was written using Glockenspiel C++ version 2.0a.

B. PRINTING NOTES

This code was prepared for printout using the `c++2latex` code generator. This generator parses the ASCII text input files and places latex commands where directed. This makes the code layout more readable and highlights the important data structures and key words within the C++ code. The latex output was then converted to postscript format with the `dvitops` program.

C. MAINTENANCE

This code is maintained at the Naval Postgraduate School Computer Science department.

```

// File Header -----
//.....:
//.Filename.....: main.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char mainSecsId[] = "@(#)main.cxx 1.3\t9/16/91";

// Interface Dependencies -----

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <Object.h>
#include <Transaction.h>
#include <Directory.h>
#include <GlobalEntities.h>
#include <Database.h>
#include <List.h>
#include <stream.hxx>

extern "C--"
{
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
}

#ifndef __VOBJECTFUNC_H
#include "vobjectfunc.h"
#endif

#ifndef __EVALUATION_H
#include "evaluation.h"
#endif

#ifndef __PROTFUNC_H
#include "protfunc.h"
#endif

```

```

#ifndef _CONFFUNC_H
#include "conffunc.h"
#endif

// End Interface Dependencies -----

Type *V_OBJECT_OType;
Type *THREAD_OType;
Type *COMPONENT_OType;
Type *TEXT_OBJECT_OType;
Type *PROTOTYPE_OType;
Type *CONFIGURATION_OType;
Type *DDB_OType;

Directory *prototype_dir = (Directory*)0;
Directory *ddbRootDir = (Directory*)0;
List *myPrototypeList = (List*)0;
char *database_name = (char *)0;
char *userPtr = (char *)0;
char *dirNamePtr = (char *)0;
void initialize_types(void);
void setUpDirectory(char *, char *);

int main(int argc, char *argv[])
{
    dirNamePtr = getenv("PROTOTYPE");
    userPtr = getenv("USER");
    Boolean done=FALSE;
    int menu_option=0;
    ifstream inFile;

    char *function_tag;

    int number_arguments;
    int run = 0;

    if (argv[1])
    {
        database_name = new char[strlen(argv[1])+1];
        strcpy(database_name,argv[1]);
    }

    if (argv[2])
    {
        function_tag = new char[strlen(argv[2])+1];
        strcpy(function_tag, argv[2]);
    }

    number_arguments = argc - 3;

```

```

// subtract the "design" "database" & "function"
// from the argument(s) we evaluate.

if (number_arguments < 0)
{
    cerr << "<ERROR: Not enough arguments specified>\n\n"
        << " Format for Usage:  \n\n"
        << " designdb dbname function [argument, ...]\n\n";
    exit(1);
}

if (OC.open(database_name))
{
    OC.transactionStart();

    initialize_types();

    setUpDirectory(argv[3], function_tag);
}
else
{
    cerr << "<ERROR: Error attempting to open database "
        << database_name
        << " ---> database does not exist in registry\n\n\n";
    exit(1);
}

if (strlen(function_tag)<3 || strlen(function_tag)>3)
{
    cerr << "<ERROR: illegal function type--> <"
        << function_tag << "> >\n";
}
else
{
    if (function_tag[0] == 'P' || function_tag[0]=='p')
        run = evaluate_prototype_function(upper(function_tag),
                                           number_arguments);
    else if (function_tag[0] == 'C' || function_tag[0]=='c')
        run=evaluate_configuration_function(upper(function_tag),
                                           number_arguments);
    else if (function_tag[0] == 'V' || function_tag[0]=='v')
        run = evaluate_vobject_function(upper(function_tag),
                                         number_arguments);
    else
    {
        cerr << "<ERROR: illegal function type--> <";
        cerr << function_tag << "> >\n";
    }
}

switch (run)

```

```

{
case LIST_PROTOTYPES:
    list_prot_func(number_arguments);
    break;
case LONG_LIST_PROTOTYPES:
    long_list_prot_func(number_arguments);
    break;
case GET_PROTOTYPE_LEADER:
    get_prot_leader_func(number_arguments, argv[3]);
    break;
case DUMP_PROTOTYPE_SUMMARY:
    dump_prot_summary_func(number_arguments, argv[3]);
    break;
case GET_PROTOTYPE_DESCRIPTION:
    get_prot_description_func(number_arguments, argv[3]);
    break;
case RETRIEVE_PROTOTYPE_DATE:
    retrieve_prot_date_func(number_arguments, argv[3]);
    break;
case INSERT_PROTOTYPE:
    insert_prot_func(number_arguments, argv[3],
                    argv[4], argv[5]);

    break;
case UPDATE_PROTOTYPE_LEADER:
    update_prot_leader_func(number_arguments,
                          argv[3], argv[4]);

    break;
case UPDATE_PROTOTYPE_DESC:
    update_prot_desc_func(number_arguments,
                        argv[3], argv[4]);

    break;
case UPDATE_PROTOTYPE_NAME:
    update_prot_name_func(number_arguments,
                        argv[3], argv[4]);

    break;
case DUMP_CONFIGURATION_SUMMARY:
    dump_conf_summary_func(number_arguments,
                        argv[3], argv[4]);

    break;
case GET_LATEST_CONFIGURATION:
    get_latest_conf_func(number_arguments,
                        argv[3]);

    break;
case ADD_CONFIGURATION_OPERATORS:
    add_conf_operators_func(number_arguments,
                        argv[3], argv[4]);

    break;
case DUMP_CONFIGURATION_OPERATORS:
    dump_conf_operators_func(number_arguments,
                        argv[3], argv[4], argv[5]);

    break;

```



```

case RELEASE_CONFIGURATION_LOCK:
    release_conf_lock_func(number_arguments,
                           argv[3],argv[4]);

    break;
case LONG_LIST_CONFIGURATION_OPERATORS:
    long_list_conf_operators_func(number_arguments,
                                   argv[3],argv[4]);

    break;
case LIST_CONFIGURATION_OPERATORS:
    list_conf_operators_func(number_arguments,
                              argv[3],argv[4]);

    break;
case LIST_CONFIGURATION_DEFAULT_OPERATOR:
    list_conf_default_operator_func(number_arguments,
                                     argv[3],argv[4]);

    break;
case LIST_CONFIGURATIONS:
    list_conf_func(number_arguments, argv[3]);
    break;
case UPDATE_CONFIGURATION_NAME:
    update_conf_name_func(number_arguments,
                          argv[3], argv[4],
                          argv[5]);

    break;
case GET_CONFIGURATION_DESCRIPTION:
    get_conf_desc_func(number_arguments,
                       argv[3], argv[4]);

    break;
case INSERT_CONFIGURATION:
    insert_conf_func(number_arguments, argv[3],
                    argv[4], argv[5], argv[6]);

    break;
case UPDATE_CONFIGURATION_DESCRIPTION:
    update_conf_desc_func(argv[3], argv[4],
                          argv[5]);

    break;
case GET_CONFIGURATION_MANAGER:
    get_conf_manager_func(number_arguments,
                          argv[3], argv[4]);

    break;
case UPDATE_CONFIGURATION_MANAGER:
    update_conf_manager_func(number_arguments, argv[3],
                             argv[4], argv[5]);

    break;
case GET_CONFIGURATION_DATE:
    get_conf_date_func(number_arguments,
                      argv[3], argv[4]);

    break;
case GET_CONFIGURATION_CHANGED:

case POST_CONFIGURATION_LOG:

```

```

    post_conf_log_func(argv[3], argv[4], argv[5]);
    break;
case GET_CONFIGURATION_LOG:
    get_conf_log_func(number_arguments,
                      argv[3], argv[4]);
    break;
case ATTACH_OPERATOR:
    attach_vobject_to_conf_func(number_arguments,
                                argv[3], argv[4],
                                argv[5], argv[6]);
    break;
// ---X-----X-----X-----X
//
// VOBJECT CASE STATEMENTS
//
// ---X-----X-----X-----X
case LIST_OPERATORS:
    list_operators_func(number_arguments, argv[3],
                       argv[4], argv[5]);
    break;
case GET_VOBJECT_DESCRIPTION:
    get_vobject_desc_func(number_arguments, argv[3],
                          argv[4], argv[5]);
    break;
case UPDATE_VOBJECT_DESCRIPTION:
    update_vobject_desc_func(number_arguments,
                              argv[3], argv[4],
                              argv[5], argv[6]);
    break;
case GET_VOBJECT_DATE:
    get_vobject_date_func(number_arguments, argv[3],
                          argv[4], argv[5]);
    break;
case GET_VOBJECT_VERSIONS:
    get_vobject_versions_func(number_arguments,
                              argv[3], argv[4]);
    break;
case GET_VOBJECT_LOCK:
    get_vobject_lock_func(number_arguments,
                          argv[3], argv[4], argv[5]);
    break;
case GET_VOBJECT_VERSION:
    get_vobject_version_func();
    break;
case DUMP_VOBJECT_SUMMARY:
    dump_vobject_summary_func(number_arguments, argv[3],
                              argv[4], argv[5]);
    break;
case GET_VOBJECT_POSTSCRIPT:
    get_vobject_psfile_func(number_arguments, argv[3],
                            argv[4], argv[5], argv[6]);

```

```

    break;
case GET_VOBJECT_GRAPH:
    get_vobject_graphfile_func(number_arguments,argv[3],
                                argv[4], argv[5],argv[6]);

    break;
case GET_VOBJECT_IMPLEMENTATION:
    get_vobject_impfile_func(number_arguments, argv[3],
                                argv[4], argv[5],argv[6]);

    break;
case GET_VOBJECT_SPECIFICATION:
    get_vobject_specfile_func(number_arguments, argv[3],
                                argv[4], argv[5],argv[6]);

    break;
case GET_VOBJECT_SOURCE:
    get_vobject_sourcefile_func(number_arguments,
                                argv[3], argv[4],
                                argv[5],argv[6]);

    break;
case DUMP_VOBJECT_FILES:
    dump_vobject_files_func(number_arguments,
                                argv[3], argv[4],
                                argv[5],argv[6]);

    break;
case DUMP_VOBJECT_TREE:
    dump_vobject_tree_func(number_arguments, argv[3],
                                argv[4], argv[5],argv[6]);

    break;
case ADD_VOBJECT_AND_SUBTREE:
    add_vobject_and_subtree_func(number_arguments,
                                argv[3],argv[4]);

    break;
case RELEASE_OPERATOR_LOCK:
    release_operator_lock_func(number_arguments,
                                argv[3],argv[4],
                                argv[5]);

    break;
case RELEASE_SUBTREE_LOCK:
    release_subtree_lock_func(number_arguments,
                                argv[3],argv[4],
                                argv[5]);

    break;
case LONG_LIST_OPERATORS:
    long_list_operators_func(number_arguments,argv[3],
                                argv[4],argv[5]);

    break;
case LONG_LIST_CHILDREN:
    long_list_children_func(number_arguments,argv[3],
                                argv[4],argv[5]);

    break;
case LONG_LIST_PARENTS:
    long_list_parents_func(number_arguments,argv[3].

```

```

        argv[4],argv[5]);
    break;
case ERROR_IN_EVALUATION:
    cerr << "<ERROR: Error returned from evaluation"
        << " function>\n";
    break;
default:
    {
        cerr << "<ERROR: Unknown error with function "
            << " call..switch (run)...>\n";
        exit(1);
    }
}
OC_transactionCommit();
OC_close(database_name);
cerr << "\n\nDesign Database "
    << "v1.1 Copyright 1991(c) WP.D.A.L.G Inc.\n\n\n";
exit(0);
}

// -----
// END OF MAIN
//
// -----

void initialize_types(void)
{
    THREAD_OType=(Type *)OC_lookup("THREAD");
    COMPONENT_OType=(Type *)OC_lookup("COMPONENT");
    V_OBJECT_OType=(Type *)OC_lookup("V_OBJECT");
    TEXT_OBJECT_OType=(Type *)OC_lookup("TEXT_OBJECT");
    PROTOTYPE_OType=(Type *)OC_lookup("PROTOTYPE");
    CONFIGURATION_OType = (Type *)OC_lookup("CONFIGURATION");
    DDB_OType = (Type *)OC_lookup("DDB");
};

void setUpDirectory(char *protName, char *func_tag)
{
    ddbRootDir = (Directory *) OC_lookup(DSIGN_DATABASE_DIRECTORY);
    if(ddbRootDir)
    {
        OC_setWorkingDirectory(ddbRootDir);
        myPrototypeList = (List *) OC_lookup(PROTOTYPE_LIST);
    }
    else
    {
        ddbRootDir = new Directory(DSIGN_DATABASE_DIRECTORY);
        ddbRootDir -> putObject();
        OC_setWorkingDirectory(ddbRootDir);
        myPrototypeList = new List(OC_string.PROTOTYPE_LIST);
    }
}

```

```

        myPrototypeList → putObject();
    }
    char *yourDirChoice;
    if ((strcmp(func_tag, LIST_PROTOTYPE_UPC) == 0) ||
        (strcmp(func_tag, LIST_PROTOTYPE_LC) == 0) ||
        (strcmp(func_tag, LONG_LIST_PROTOTYPE_LC) == 0) ||
        (strcmp(func_tag, LONG_LIST_PROTOTYPE_LC) == 0))
    {
        return;
    }
    else
    {
        yourDirChoice = new char[1 + strlen(protName) + 5];
        strcpy(yourDirChoice, protName);
        strcat(yourDirChoice, "_dir");
    }
    prototype_dir = (Directory *)OC_lookup(yourDirChoice);

    if (((!strcmp(func_tag, INSERT_PROTOTYPE_UPC)) == 0) &&
        (!strcmp(func_tag, INSERT_PROTOTYPE_LC)) == 0) &&
        (!prototype_dir))
    {
        cerr << "<ERROR: Prototype " << protName
            << " not found>\n"
            << "<Did you remember to run "
            << "INSERT PROTOTYPE first?>\n";
        OC_transactionCommit();
        OC_close(database_name);
        exit(0);
    }
    if (!prototype_dir)
    {
        prototype_dir = new Directory(yourDirChoice);
        prototype_dir → putObject();
        if (!prototype_dir)
            cerr << "Did not setup database directory\n";
        OC_setWorkingDirectory(prototype_dir);
    }
    else
    {
        OC_setWorkingDirectory(prototype_dir);
    }
}

```

```

// File Header -----
//.....:
//.Filename.....: component.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _COMPONENT_H
#define _COMPONENT_H

    // SCCS ID follows: will compile to place date/time stamp in
    // object file

    static char COMPONENT_h_ScscId[] = "@(#)component.h 1.3\t9/16/91";

// Contents -----
//
// COMPONENT
//
// Description
//
// Defines class COMPONENT.
//
// End -----

// Implementation Dependencies -----

#include <Type.h>
#include <List.h>
#include <Reference.h>
#include "ReferenceMacros.h"
#include <stream.hxx>

    // End Implementation Dependencies -----

// Interface Dependencies -----

#ifndef _TEXT_OBJECT_H
#include "text_object.h"
#endif

// End -----

    TypeCheckReference(TextObjDictReference, Reference, List);

// Class //

```

```

class COMPONENT : public Object
{
private :

    TextObjDictReference text_object_list;

public:

    COMPONENT();
    COMPONENT(APL *theAPL);
    virtual void Destroy(Boolean abort = FALSE);
    virtual Type *getDirectType();
    void getComponentNames();
    Boolean getComponentSource(char*);
    void addTextObject(TEXT_OBJECT *);
    Boolean getPSfile(char*);
    Boolean getGRAPHfile(char*);
    Boolean getSPECfile(char*);
    Boolean getIMPfile(char*);
    Boolean getSOURCEfile(char*);
    ~COMPONENT() { Destroy(FALSE); };
};

```

```

// Description -----
//
// Defines an COMPONENT class. The class COMPONENT is a derived
// class of Object.
//
// Constructor
//
// COMPONENT
//
// Constructs an COMPONENT object
//
// COMPONENT -APL
//
// ONTOS required constructor
//
// Public Members
//
// Destroy
//
// ONTOS heap mangagement method.
//
// getDirectType
//
// Return the ONTOS Type of class COMPONENT.
//
// GetComponentNames
//

```

```

// Display the file names of the file contained in the COMPONENT node
//
// GetComponentSource
//
// Output the contents of an COMPONENT node to files.
//
// addTextObject
//
// Inserts a text_object into the COMPONENT node.
//
// getPSfile
//
// Output the .ps file contained in the COMPONENT node.
//
// getGRAPHfile
//
// Output the .graph file contained in the COMPONENT node.
//
// getSPECfile
//
// Output the .spec file contained in the COMPONENT node.
//
// getIMPfile
//
// Output the .imp file contained in the COMPONENT node.
//
// getSourcefile
//
// Output the .a file contained in the COMPONENT node.
//
// ~COMPONENT
//
// Destructor for the COMPONENT class.
//
// End -----
#endif // _COMPONENT_H

```



```

// File Header -----
//.....:
//.Filename.....: component.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char COMPONENT_cxx_SccsId[] = "@(#)component.cxx 1.3\t9/16/91";

// Contents -----
//
// COMPONENT::COMPONENT ONTOS constructor
// COMPONENT::COMPONENT constructor
// COMPONENT::getDirectType
// COMPONENT::Destroy
// COMPONENT::GetComponentNames
// COMPONENT::GetComponentSource
// COMPONENT::addTextObject
// COMPONENT::getPSfile
// COMPONENT::getGRAPHfile
// COMPONENT::getSPECfile
// COMPONENT::getIMPfile
// COMPONENT::getSOURCEfile
//
// Description
//
// Implementation of class COMPONENT member functions.
//
// End -----

// Implementation Dependencies -----

#ifndef _DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <Object.h>
#include <GlobalEntities.h>

extern "C--"
{
#include <strings.h>
}

```

```

#ifndef _TRACER.H
#include "tracer.h"
#endif

#ifndef _COMPONENT.H
#include "component.h"
#endif

// End Implementation Dependencies-----

extern Type *COMPONENT_OType;
extern Type *TEXT_OBJECT_OType;

// ONTOS required constructor //

COMPONENT::COMPONENT(APL *theAPL):(theAPL)
{
};

// Constructor //

COMPONENT::COMPONENT()

    // Summary -----
    //
    // Constructs a persistent COMPONENT object.
    //
    // Parameter
    //
    // None
    //
    // Functional description
    //
    // Creates a list to hold text objects, then reset a reference
    // to point to the list.
    //
    // End -----
{
    initDirectType(COMPONENT_OType);

    List *newTextObjList = new List(TEXT_OBJECT_OType);
    newTextObjList → putObject();
    text_object_list.Reset(newTextObjList, this);
    putObject();
};

// End Constructor COMPONENT::COMPONENT//

// Member Function //

Type *COMPONENT::getDirectType()

```

```

// Summary -----
//
// returns the ONTOS Type for the COMPONENT class.
//
// Return value
//
// A pointer to an ONTOS Type.
//
// End -----

{
    return COMPONENT_OType;
};

// End Member Function COMPONENT::getDirectType //

// Member Function //

void COMPONENT::Destroy(Boolean aborted)
{
    if(aborted)
    {
        Object::Destroy(aborted);
    }
}

// End Member Function COMPONENT::Destroy -----

// Member Function //

void COMPONENT::GetComponentNames()

    // Summary -----
    //
    // Displays the name of each component in an COMPONENT object.
    //
    // Parameter
    //
    // None
    //
    // Functional description
    //
    // We get a pointer to the list and then create an iterator
    // for the list. We iterate over each text object and
    // display the contents of the name field.
    //
    //
    // End -----

{
    List *my_list = (List*)text_object_list.Binding(this);

```

```

ListIterator my_iterator(my_list);
TEXT_OBJECT *the_text_object;

while(my_iterator.moreData())
{
    the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
    the_text_object → displayFileName();
}

// End Member Function COMPONENT::getComponentNames -----

// Member Function //

Boolean COMPONENT::getComponentSource(char *fileMode)
{
    List *my_list = (List*)text_object_list.Binding(this);
    ListIterator my_iterator(my_list);
    TEXT_OBJECT *the_text_object;
    Boolean write_failed = FALSE;
    while(my_iterator.moreData())
    {
        the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
        if (!the_text_object → rebuildTextFile(fileMode))
            write_failed = TRUE;
    }
    if (write_failed)
        return FAILED;
    else
        return SUCCESS;
}

// End Member Function COMPONENT::getComponentSource -----

// Member Function //

void COMPONENT::addTextObject(TEXT_OBJECT *my_text_object)
{
    List *my_list = (List*)text_object_list.Binding(this);
    my_list → Insert(my_text_object);
    my_list → putObject();
    putObject();
}

// End Member Function COMPONENT::addTextObject -----

// Member Function //

Boolean COMPONENT::getPSfile(char *fileMode)
{
    List *my_list = (List*)text_object_list.Binding(this);

```

```

ListIterator my_iterator(my_list);

while(my_iterator.moreData())
{
    TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
    char *the_file_name = the_text_object->getFileName();
    the_file_name=(the_file_name +
        (strlen(the_text_object->getFileName())-LENGTH_PS_EXT));
    if(strcmp(the_file_name,PS_EXT)==0)
    {
        if (the_text_object->rebuildTextFile(fileMode));
        return SUCCESS;
    }
}

// End Member Function COMPONENT::getPSfile -----

// Member Function //

Boolean COMPONENT::getSPECfile(char *fileMode)
{
    List *my_list = (List*)text_object_list.Binding(this);
    ListIterator my_iterator(my_list);
    while(my_iterator.moreData())
    {
        TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
        char *the_file_name = the_text_object->getFileName();
        the_file_name=(the_file_name +
            (strlen(the_text_object->getFileName())-LENGTH_SPEC_EXT));
        if(strcmp(the_file_name,SPEC_EXT)==0)
        {
            if (the_text_object->rebuildTextFile(fileMode))
                return SUCCESS;
            else
                return FAILED;
        }
    }
}

// End Member Function COMPONENT::getSPECfile -----

// Member Function //

Boolean COMPONENT::getGRAPHfile(char *fileMode)
{
    List *my_list = (List*)text_object_list.Binding(this);
    ListIterator my_iterator(my_list);
    while(my_iterator.moreData())
    {

```

```

TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
char *the_file_name = the_text_object->getFileName();
the_file_name=(the_file_name +
                (strlen(the_text_object->getFileName())-LENGTH_GRAPH_EXT));
if(strcmp(the_file_name,GRAPH_EXT)==0)
{
    if (the_text_object->rebuildTextFile(fileMode))
        return SUCCESS;
    else
        return FAILED;
}
}
}

```

// End Member Function COMPONENT::getGRAPHfile -----

// Member Function //

```

Boolean COMPONENT::getIMPfile(char *fileMode)
{
    List *my_list = (List*)text_object_list.Binding(this);
    ListIterator my_iterator(my_list);
    while(my_iterator.moreData())
    {
        TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
        char *the_file_name = the_text_object->getFileName();
        the_file_name=(the_file_name +
                        (strlen(the_text_object->getFileName())-LENGTH_IMP_EXT));
        if(strcmp(the_file_name,IMP_EXT)==0)
        {
            if (the_text_object->rebuildTextFile(fileMode))
                return SUCCESS;
            else
                return FAILED;
        }
    }
}

```

// End Member Function COMPONENT::getIMPfile -----

// Member Function //

```

Boolean COMPONENT::getSOURCEfile(char *fileMode)
{
    List *my_list = (List*)text_object_list.Binding(this);
    ListIterator my_iterator(my_list);
    while(my_iterator.moreData())
    {
        TEXT_OBJECT *the_text_object = (TEXT_OBJECT*)(Entity*)my_iterator();
        char *the_file_name = the_text_object->getFileName();
        the_file_name=(the_file_name +

```

```

        (strlen(the_text_object->getFileName())-LENGTH_SOURCE_EXT));
if(strcmp(the_file_name,SOURCE_EXT)==0)
{
    if (the_text_object->rebuildTextFile(fileMode))
        return SUCCESS;
    else
        return FAILED;
}
}
// End Member Function COMPONENT::getSourcefile -----

```

```

// File Header -----
//.....:
//.Filename.....: conffunc.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifdef _CONFFUNC_H
#define _CONFFUNC_H

    // SCCS ID follows: will compile to place date/time stamp in
    // object file

    static char conffunc_h_ScCsId[] = "0(#)conffunc.h 1.3\t9/16/91";

// Contents -----
//
// Prototypes for functions related to manipulating
// configurations.
//
// End -----

    void list_conf_func(int, char *);
    void dump_conf_summary_func(int, char *, char *);
    void get_latest_conf_func(int, char *);
    void attach_vobject_to_conf_func(int, char *, char *, char *, char *);
    void update_conf_name_func(int, char*, char*, char*);
    void get_conf_desc_func(int, char*, char*);
    void insert_conf_func(int, char*, char*, char*, char*);
    void update_conf_desc_func(char*, char*, char*);
    void get_conf_manager_func(int, char*, char*);
    void update_conf_manager_func(int, char*, char*, char*);
    void get_conf_date_func(int, char*, char*);
    void post_conf_log_func(char*, char*, char*);
    void get_conf_log_func(int, char*, char*);
    void dump_conf_operators_func(int, char *, char *, char *);
    void add_conf_operators_func(int, char *, char *);
    void release_conf_lock_func(int, char *, char *);
    void long_list_conf_operators_func(int, char *, char *);
    void list_conf_operators_func(int, char *, char *);
    void list_conf_default_operator_func(int, char *, char *);

// Description -----
//
// list_conf_func
//

```



```

// Provides a list of configurations associated with a
// designated prototype.
//
// dump_conf_summary_func
//
// Provides summary information of a configuration to
// to include: creation date, manager, version number and
// default VOBJECT name, and a description.
//
// get_latest_conf_func
//
// Provides the name of the most recently added configuration.
//
// attach_vobject_to_conf_func
//
// Attaches an instance of the VOBJECT class to a configuration.
//
// update_conf_name_func
//
// Changes the name of a designated configuration.
//
// get_conf_desc_func
//
// Provides a description of the designated configuration.
//
// insert_conf_func
//
// Creates a new configuration and binds it to a particular
// prototype.
//
// update_conf_desc_func
//
// Updates the description text of a designated configuration.
//
// get_conf_manager_func
//
// Provides a configuration manager's name.
//
// update_conf_manager_func
//
// Changes a configuration manager's name.
//
// get_conf_date_func
//
// Provides the creation date of a given configuration.
//
// post_conf_log_func
//
// Adds a timestamped log entry (translated char string)
// to a configuration.
//

```

```

// get_conf_log_func
//
// Provides the text of a configuration log.
//
// dump_conf_operators_func
//
// Write a copy of the operators in a designated configuration
// to disk.
//
// add_conf_operators_func
//
// Add operator from disk to a particular configuration in the
// database.
//
// release_conf_lock_func
//
// Provide a method to release locked operators in a given
// configuration.
//
// long_list_conf_operators_func
//
// List all the children's names and version numbers of a
// given configuration.
//
// list_conf_operators_func
//
// List the immediate children's name and version number of
// a given configuration.
//
// list_conf_default_operator_func
//
// Provides the name and version number of the default VOBJECT
// assigned to the configuration.
//
// End Description -----
#endif // _CONFFUNC_H

```

```

// File Header -----
//.....:
//.Filename.....: conffunc.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char conffunc_cxx_SccsId[] = "@(#)conffunc.cxx 1.3\t9/16/91";

// Contents -----
//
// list_conf_func
// dump_conf_summary_func
// get_latest_conf_func
// attach_vobject_to_conf_func
// update_conf_name_func
// get_conf_desc_func
// insert_conf_func
// update_conf_desc_func
// get_conf_manager_func
// update_conf_manager_func
// get_conf_date_func
// post_conf_log_func
// get_conf_log_func
// dump_conf_operators_func
// add_conf_operators_func
// release_conf_lock_func
// long_list_conf_operators_func
// list_conf_operators_func
// list_conf_default_operator_func
//
// End -----

// Implementation Dependencies -----

#include <stream.hxx>
#include <Directory.h>

extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
#include <stdlib.h>

```

```
}
```

```
#ifndef _DIRECTORY_H
#include "directory.h"
#endif
```

```
#ifndef _TREE_H
#include "tree.h"
#endif
```

```
#ifndef _TREENODE_H
#include "treenode.h"
#endif
```

```
#ifndef _PROTOTYPE_H
#include "prototype.h"
#endif
```

```
#ifndef _THREAD_H
#include "thread.h"
#endif
```

```
#ifndef _CONFIGURATION_H
#include "configuration.h"
#endif
```

```
#ifndef _CONFFUNC_H
#include "conffunc.h"
#endif
```

```
#ifndef _DDBDEFINES_H
#include "ddbdefines.h"
#endif
```

```
PROTOTYPE *protoPtr;
CONFIGURATION *configurationPtr;
extern THREAD *threadPtr;
```

```
// End Implementation Dependencies -----
```

```
ifstream inputfile;
```

```
void list_conf_func(int number_arguments, char *arg1)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 1:
```

```

        protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
        protoPtr → listConfigurations();
        break;
    case 3:
    default:
        cerr << "<ERROR: extra arguments in list configurations>\n";
    }
}

void dump_conf_summary_func(int number_arguments, char *arg1, char *arg2)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch(number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION*)OC_lookup(arg2);
                if(configurationPtr)
                {
                    configurationPtr → dumpConfigSummary();
                }
                else
                {
                    cerr << "<ERROR: " << arg2 << " configuration not found>\n";
                    cerr << "<Dump configuration operation terminated>\n";
                    return;
                }
            }
            else
            {
                cerr << "<ERROR: " << arg1 << " prototype not found>\n";
                cerr << " Dump configuration operation terminated\n";
                return;
            }
            break;
        default:
            cerr << "<ERROR: extra arguments in dump configuration summary>\n";
    }
}

void get_latest_conf_func(int number_arguments, char *arg1)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

```

```

switch (number_arguments)
{
    case 1:
        protoPtr = (PROTOTYPE*)OC_lookup(proto_name);
        protoPtr → getDefaultConfigName();
        break;
    case 3:
    default:
        cerr << "<ERROR: extra arguments in get default configurations>\n";
}
}

void attach_vobject_to_conf_func(int number_arguments, char *proto_name,
                                char *config_name, char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 3:
            protoPtr = (PROTOTYPE*)OC_lookup(proto_name);
            if (protoPtr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    configurationPtr = (CONFIGURATION *)OC_lookup(config_name);
                    if (configurationPtr)
                    {
                        V_OBJECT *vobjectPtr;
                        vobjectPtr = threadPtr→current();
                        configurationPtr→attachVobjecttoConfig(vobjectPtr);
                        configurationPtr→putObject();
                    }
                    else
                    {
                        cerr << "<Error getting configuration in attach vobject to config>\n";
                    }
                }
                else
                {
                    cerr << "<Error getting thread in ATTACH_VOBJECT_TO_CONFIG:>\n";
                }
            }
            else
            {
                cerr << "<Error getting Prototype in ATTACH_VOBJECT_TO_CONFIG:>\n";
            }
        break;
    }
}

```

```

case 4:
    protoPtr = (PROTOTYPE*)OC_lookup(proto_name);
    if (protoPtr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            configurationPtr = (CONFIGURATION *)OC_lookup(config_name);
            if (configurationPtr)
            {
                V_OBJECT *vobjectPtr;
                vobjectPtr = threadPtr->version(atoi(versionstr));
                configurationPtr->attachVobjecttoConfig(vobjectPtr);
                configurationPtr->putObject();
            }
            else
            {
                cerr << "<Error getting configuration in attach vobject to config>\n";
            }
        }
        else
        {
            cerr << "<Error getting thread in ATTACH_VOBJECT_TO_CONFIG:>\n";
        }
    }
    else
    {
        cerr << "<Error getting Prototype in ATTACH_VOBJECT_TO_CONFIG:>\n";
    }
    break;
default:
    cerr << "<ERROR: invalid number args for get vobject description>\n";
}
}

```

```

void update_conf_name_func(int number_arguments, char *arg1, char *arg2, char *arg3)
{
    char *proto_name = new char [strlen(arg1)+5];
    strcpy(proto_name,arg1);
    strcat(proto_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 3:
            protoPtr = (PROTOTYPE*)OC_lookup(proto_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION*)OC_lookup(arg2);
                if (configurationPtr)
                {

```

```

        configurationPtr → updateConfigName(arg3);
    }
    else
    {
        cerr << "<Prototype " << arg1 << "does not "
            << "contain configuration " << arg2 << ">\n"
            << "<Update configuration Name operation aborted.>\n";
        break;
    }
}
else
{
    cerr << "<Prototype " << arg1 << " not found>\n"
        << "<update configuration name operation aborted>";
}
break;
default:
    cerr << "<ERROR: invalid number args for update Config Name>\n";
}
}

```

```

void get_conf_desc_func(int number_arguments, char *arg1, char *arg2)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION *)OC_lookup(arg2);
                if (configurationPtr)
                {
                    configurationPtr → getConfigDescription();
                }
                else
                {
                    cerr << "<Configuration: " << arg2 << " is not contained "
                        << "in prototype " << arg1 << ">\n"
                        << "<get configuration Description Operation Aborted.>\n";
                }
            }
        else
        {
            cerr << "<Prototype " << arg1 << " not found>\n"
                << "<get configuration description operation aborted>";
        }
    }
}

```



```

        break;
    default:
        cerr << "<ERROR: invalid number args for get configuration description>\n";
    }
}

```

```

void insert_conf_func(int number_arguments, char *arg1, char *arg2, char *arg3, char *arg4)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION *)OC_lookup(arg2);
                if (!configurationPtr)
                {
                    configurationPtr = new CONFIGURATION(arg2);
                    protoPtr → addConfiguration(configurationPtr);
                }
            }
            else
            {
                cerr << "<ERROR: Configuration name " << arg2 << " already exists!>\n";
                return;
            }
        }
        else
        {
            cerr << "<Prototype " << arg1 << " not found>\n"
                << "<no Configuration operation conducted>";
        }
        break;
    case 3:
        protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
        if (protoPtr)
        {
            configurationPtr = (CONFIGURATION *)OC_lookup(arg2);
            if (!configurationPtr)
            {
                configurationPtr = new CONFIGURATION(arg2, arg3);
                protoPtr → addConfiguration(configurationPtr);
            }
        }
        else
        {
            cerr << "<ERROR: Configuration name " << arg2 << " already exists!>\n";
            return;
        }
    }
}

```

```

    }
}
else
{
    cerr << "<Prototype " << arg1 << " not found>\n"
        << "<no Configuration operation conducted>";
}
break;
case 4:
    protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (protoPtr)
    {
        configurationPtr = (CONFIGURATION *)OC_lookup(arg2);
        if (!configurationPtr)
        {
            configurationPtr = new CONFIGURATION(arg2, arg3);
            inputfile.open(arg4, ios::in);
            if (!inputfile)
            {
                cerr << "<File with config description contents does not exist>\n"
                    << "<Constructing configuration w/Name & Manager only>\n";
            }
            else
            {
                configurationPtr -> updateConfigDescription(arg4, inputfile);
            }
            protoPtr -> addConfiguration(configurationPtr);
            inputfile.close();
        }
        else
        {
            cerr << "<ERROR: Configuration name " << arg2 << " already exists!>\n";
            return;
        }
    }
    else
    {
        cerr << "<Prototype " << arg1 << " not found>\n"
            << "<no Configuration operation conducted>";
    }
    break;
default:
    cerr << "<ERROR: invalid number args for insert configuration>\n";
}
}

```

```

void update_conf_desc_func(char *arg1, char *arg2, char *arg3)
{
    char *prototype_name = new char [strlen(arg1)+5];

```

```

strcpy(prototype_name,arg1);
strcat(prototype_name,PROTOTYPE_EXT);

protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
if (protoPtr)
{
    configurationPtr = (CONFIGURATION*)OC_lookup(arg2);
    if (configurationPtr)
    {
        inputfile.open(arg3, ios::in);
        if (!inputfile)
        {
            cerr << "<File with config description contents not found>\n"
                << "<Aborting update configuration operation>\n";
        }
        else
        {
            configurationPtr -> updateConfigDescription(arg3, inputfile);
            inputfile.close();
        }
    }
    else
    {
        cerr << "<Prototype " << arg1 << "does not "
            << "contain configuration " << arg2 << ">\n"
            << "<Update configuration Description operation aborted.>\n";
    }
}
else
{
    cerr << "<Prototype " << arg1 << " not found>\n"
        << "<no Configuration operation conducted>";
}
}

```

```

void get_conf_manager_func(int number_arguments, char *arg1, char *arg2)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION *)OC_lookup(arg2);
                if (configurationPtr)
                {

```

```

        configurationPtr → getConfigManager();
    }
    else
    {
        cerr << "<Configuration: " << arg2 << " is not contained "
            << "in prototype " << arg1 << ".>\n"
            << "<get configuration Manager Operation Aborted.>\n";
    }
}
else
{
    cerr << "<Prototype " << arg1 << " not found>\n"
        << "<get configuration manager operation aborted>";
}
break;
default:
    cerr << "<ERROR: invalid number args for insert configuration>\n";
}
}

```

```

void update_conf_manager_func(int number_arguments, char *arg1, char *arg2, char *arg3)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 3:
            protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION*)OC_lookup(arg2);
                if (configurationPtr)
                {
                    configurationPtr → updateConfigManager(arg3);
                }
                else
                {
                    cerr << "<Prototype " << arg1 << "does not "
                        << "contain configuration " << arg2 << ">\n"
                        << "<Update configuration Manager operation aborted.>\n";
                }
            }
        else
        {
            cerr << "<Prototype " << arg1 << " not found>\n"
                << "<update configuration Manager operation aborted>";
        }
    }
    break;
}

```

```

        default:
            cerr << "<ERROR: invalid number args for update Config Manager>\n";
        }
    }

void get_conf_date_func(int number_arguments, char *arg1, char *arg2)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_Lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION*)OC_Lookup(arg2);
                if (configurationPtr)
                {
                    time_t systemtime = configurationPtr -> getConfCreationDate();
                    cout << ctime(&systemtime) << "\n";
                }
                else
                {
                    cerr << "<Configuration " << arg2 << " not found>\n"
                        << "<no Configuration operation conducted>";
                }
            }
        else
        {
            cerr << "<Prototype " << arg1 << " not found>\n"
                << "<find configuration creation date operation aborted>";
        }
        break;
        default:
            cerr << "<ERROR: invalid number args to get configuration creation date>\n";
    }
}

```

```

void post_conf_log_func(char *arg1, char *arg2, char *arg3)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    protoPtr = (PROTOTYPE*)OC_Lookup(prototype_name);
    if (protoPtr)
    {

```

```

configurationPtr = (CONFIGURATION*)OC_lookup(arg2);
if (configurationPtr)
{
    inputfile.open(arg3, ios::in);
    if (!inputfile)
    {
        configurationPtr → addtoConfigLog(arg3);
    }
    else
    {
        configurationPtr → addtoConfigLog(inputfile);
        inputfile.close();
    }
}
else
{
    cerr << "<Prototype " << arg1 << "does not "
    << "contain configuration " << arg2 << ">\n"
    << "<Update configuration Log operation aborted.>\n";
}
}
else
{
    cerr << "<Prototype " << arg1 << " not found>\n"
    << "<no Configuration operation conducted>";
}
}

```

```

void get_conf_log_func(int number_arguments, char *arg1, char *arg2)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = (CONFIGURATION *)OC_lookup(arg2);
                if (configurationPtr)
                {
                    configurationPtr → getConfigLog();
                }
                else
                {
                    cerr << "<Configuration: " << arg2 << " is not contained "
                    << "in prototype " << arg1 << ".>\n"
                    << "<get configuration log operation aborted.>\n";
                }
            }
        }
    }
}

```

```

    }
  }
  else
  {
    cerr << "<Prototype " << arg1 << " not found>\n"
    << "<get configuration log operation aborted>";
  }
}
}

```

```

void dump_conf_operators_func(int number_arguments, char *proto_name, char *conf, char
*file_write_option)
{

```

```

  char *prototype_name = new char [strlen(proto_name)+5];
  strcpy(prototype_name, proto_name);
  strcat(prototype_name, PROTOTYPE_EXT);

  switch (number_arguments)
  {
    case 3:
      protoPtr = (PROTOTYPE*)OC_Lookup(prototype_name);
      if (protoPtr)
      {
        configurationPtr = protoPtr->getConfiguration(conf);
        if (configurationPtr)
        {
          V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
          if (vobjectPtr)
          {
            long vobject_locktime = 0;
            vobject_locktime = vobjectPtr->getLockTime();
            if (vobject_locktime > 0) // prevent checkout
            {
              if (strcmp(file_write_option, "w") == 0) // change "w" to "r"
              {
                cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                << " locked by : " << vobjectPtr->getWorker()
                << " Resetting write option to read-only>\n";
                strcpy(file_write_option, "r");
              }
              cerr << "<Caution: " << vobjectPtr->getNodeName()
              << " is locked.> \n" << "Date Locked: "
              << ctime(&vobject_locktime)
              << "Subtree checked out in read-only mode\n";
            }
          }
          else
          {
            cerr << "NODENAME ---> " << vobjectPtr->getNodeName()
            << "\nVersion: " << vobjectPtr->getVersionNumber() << "\n\n";
            Boolean file_operation_successful = FALSE;
            file_operation_successful =

```

```

        vobjectPtr → checkoutCOMPONENTNode(file_write_option);
        if ((file_operation_successful) &&
            ((strcmp(file_write_option,"W")==0) ||
             (strcmp(file_write_option,"w")==0)))
        {
            vobjectPtr → setLock(); // set root lock
            vobjectPtr → setWorker();
            vobjectPtr → resetLastOpFalse();
            vobjectPtr → putObject();
        }
        if (file_operation_successful)
            vobjectPtr → dumpSubtree(file_write_option); // dump rest of tree
        else
            cerr << "<Error checking out " << vobjectPtr → getNodeName()
                << " Aborting dump.vobject.tree.func>\n";
    }
    else
        cerr << "<Error: No Vobject is attached to dump configuration>\n";
    }
    else
    {
        cerr << "<Error getting configuration in dump configuration operators:>\n";
    }
    }
    else
    {
        cerr << "<Error getting Prototype in dump configuration operators:>\n";
    }
    }
    break;
default:
    cerr << "<ERROR: invalid number args for dump configuration operators>\n";
}
}

```

```

void add_conf_operators_func(int number_arguments,char *proto_name,char *conf)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_Lookup(prototype_name);
            if (protoPtr)
            {
                configurationPtr = protoPtr → getConfiguration(conf);
                if (configurationPtr)
                {
                    V_OBJECT *vobjectPtr = configurationPtr → getDefaultVobject();

```



```

    if (vobjectPtr)
    {
        DIRECTORY *capsdirectory;
        capsdirectory = new DIRECTORY();
        char *operator_name = new char [strlen(vobjectPtr->getName()+1)];
        strcpy(operator_name,vobjectPtr->getName());
        capsdirectory->read_directory(operator_name);
        capsdirectory->updatetimestamp();
        TREENODE_Linkedlist operatorList = capsdirectory->getOperatorList();
        TREENODE *rootnode = capsdirectory->find_treenode(operator_name);
        TREENODE *tree_root = new TREENODE(rootnode,NULL);
        TREE *workingtree = new TREE(tree_root, operator_name);
        workingtree->build_tree(tree_root,operatorList);
        cerr << "CHECKIN--> " << operator_name << "\n";
        V_OBJECT *new_parent = (V_OBJECT *)0;
        new_parent = vobjectPtr->getParent();
        V_OBJECT *new_root = tree_root->checkin_node(new_parent);
        if (!new_root)
        {
            cerr << "<Error: Could not establish new_root in"
                << "add_conf_operators.func. Aborting.>\n";
            break;
        }
        new_root->setNodeName(tree_root->getname());
        tree_root->checkin_subtree(new_root);
    }
    else
        cerr << "<Error: No Vobject is attached to this configuration>\n";
}
else
{
    cerr << "<Error getting configuration in list configuration operators:>\n";
}
}
else
{
    cerr << "<Error getting Prototype in list configuration operators:>\n";
}
break;
default:
    cerr << "<ERROR: invalid number args for list configuration operators>\n";
}
}

void release_conf_lock_func(int number_arguments, char *proto_name,
                           char *conf)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);
}

```

```

switch (number_arguments)
{
case 2:
    protoPtr = (PROTOTYPE*)OC_lookup(proto_name);
    if (protoPtr)
    {
        configurationPtr = protoPtr→getConfiguration(conf);
        if (configurationPtr)
        {
            V_OBJECT *vobjectPtr = configurationPtr→getDefaultVobject();
            if (vobjectPtr)
            {
                if (vobjectPtr→releaseLock())
                {
                    vobjectPtr→putObject();
                    vobjectPtr → releaseLockSubtree();
                }
            }
            else
                cerr << "<Error: No Vobject is attached to this configuration>\n";
        }
        else
        {
            cerr << "<Error getting configuration in Release configuration Lock:>\n";
        }
    }
    else
    {
        cerr << "<Error getting Prototype in Release configuration Lock:>\n";
    }
    break;
default:
    cerr << "<ERROR: invalid number args for Release configuration lock>\n";
}
}

```

```

void long_list_conf_operators_func(int number_arguments, char *proto_name,
                                   char *conf)

```

```

{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
    case 2:
        protoPtr = (PROTOTYPE*)OC_lookup(prototype_name);
        if (protoPtr)
        {
            configurationPtr = protoPtr→getConfiguration(conf);

```

```

if (configurationPtr)
{
    V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
    if (vobjectPtr)
    {
        // cerr << "Operator: ";
        // cout << vobjectPtr->getName();
        // //*****
        // // following for loop provides spacing...
        // // *****
        // int i=0;
        // for (i=0;i<((PRINT_VERSION_LOCATION-strlen(vobjectPtr->getName())));i++)
        // cout << " ";
        // cerr << "Version: ";
        // cout << vobjectPtr->getVersionNumber();
        // cout << "\n";
        // time_t locktime = vobjectPtr->getLockTime();
        // cerr << "Locktime is: " << ctime(&locktime) << "\n";

        vobjectPtr -> longlistOperatorNames();
    }
    else
        cerr << "<Error: No Vobject is attached to this configuration>\n";
}
else
{
    cerr << "<Error getting configuration in list configuration operators:>\n";
}
}
else
{
    cerr << "<Error getting Prototype in list configuration operators:>\n";
}
break;
default:
    cerr << "<ERROR: invalid number args for list configuration operators>\n";
}
}

```

```

void list_conf_operators_func(int number_arguments, char *proto_name,
                             char *conf)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            protoPtr = (PROTOTYPE*)OC_Lookup(prototype_name);

```

```

if (protoPtr)
{
    configurationPtr = protoPtr→getConfiguration(conf);
    if (configurationPtr)
    {
        V_OBJECT *vobjectPtr = configurationPtr→getDefaultVobject();
        if (vobjectPtr)
        {
            char *name=vobjectPtr→getName();
            int version =vobjectPtr→getVersionNumber();
            cerr << "Operator:  ";
            cout << name;
            //*****
            // Added following for statement for spacing...
            // *****
            int i=0;
            for (i=0;i<(PRINT_VERSION_LOCATION -
                        strlen(vobjectPtr→getNodeName()));i++)
                cout << " ";
            cerr << "\nVersion:  ";
            cout << version << "\n";
            time_t locktime = vobjectPtr→getLockTime();
            cerr << "Locktime is:  " << ctime(&locktime) << "\n";
            vobjectPtr → listOperatorNames();
        }
        else
            cerr << "<Error:  No Vobject is attached to this configuration>\n";
    }
    else
    {
        cerr << "<Error getting configuration in list configuration operators:>\n";
    }
}
else
{
    cerr << "<Error getting Prototype in list configuration operators:>\n";
}
break;
default:
    cerr << "<ERROR:  invalid number args for list configuration operators>\n";
}
}

void list_conf_default_operator_func(int number_arguments, char *proto_name,
                                     char *conf)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)

```

```

{
case 2:
    protoPtr = (PROTOTYPE*)OC_Lookup(prototype_name);
    if (protoPtr)
    {
        configurationPtr = protoPtr->getConfiguration(conf);
        if (configurationPtr)
        {
            V_OBJECT *vobjectPtr = configurationPtr->getDefaultVobject();
            if (vobjectPtr)
            {
                char *name=vobjectPtr->getName();
                int version =vobjectPtr->getVersionNumber();
                cerr << "Operator: ";
                cout << name;
                //*****
                // Added following for statement for spacing...
                // *****
                int i=0;
                for (i=0;i<((PRINT_VERSION_LOCATION -
                    strlen(vobjectPtr->getName())));i++)
                    cout << " ";
                cerr << " Version: ";
                cout << " " << version << "\n";
            }
            else
                cerr << "<Error: No Vobject is attached to this configuration>\n";
        }
        else
        {
            cerr << "<Error getting configuration in list "
                << "configuration default operator:>\n";
        }
    }
    else
    {
        cerr << "<Error getting Prototype in list "
            << "configuration default operator:>\n";
    }
    break;
default:
    cerr << "<ERROR: invalid number args for list "
        << "configuration default operator>\n";
}
}

```



```

// File Header -----
//.....:
//.Filename.....: configuration.h
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _CONFIGURATION_H
#define _CONFIGURATION_H

    // SCCS ID follows: will compile to place date/time stamp in object file

    static char configuration_h_SccsId[] = "@(#)configuration.h 1.3\t9/16/91";

// Contents -----
//
// CONFIGURATION
//
// Description
//
// Defines class CONFIGURATION
//
// End -----

// Implementation Dependencies -----

#include <Object.h>
#include <Reference.h>
#include <Dictionary.h>
#include <stream.hxx>

    extern "C--"
    {
#include <sys/time.h>
#include <sys/types.h>
    }

#include "ReferenceMacros.h"

// End Implementation Dependencies -----

// Interface Dependencies -----

#ifndef _TEXT_OBJECT_H
#include "text_object.h"
#endif

```

```

#ifndef _VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

// End Interface Dependencies -----

TypeCheckReference(V_ObjectReference, Reference, V_OBJECT);
TypeCheckReference(LogReference, Reference, TEXT_OBJECT);
TypeCheckReference(Desc2Reference, Reference, TEXT_OBJECT);

#define DEFAULT_MANAGER ""

// Class //

class CONFIGURATION : public Object
{
private:
    char config_status;
    char *config_manager;
    time_t ConfCreationDate;
    int config_num_vobjects;
    LogReference config_log_entry;
    Desc2Reference config_description;
    V_ObjectReference theVersioned_Object;

public:
    CONFIGURATION(APL *);
    CONFIGURATION(char *name, char *manager=DEFAULT_MANAGER);
    virtual void Destroy(Boolean aborted=FALSE);
    virtual Type *getDirectType();
    void getConfigName();
    char *name();
    void getConfigStatus();
    void getConfigManager();
    void getConfigLog();
    void getConfigDescription();
    void dumpConfigSummary();
    void listConfigOperators();
    void updateConfigManager(char *new_config_manager);
    void updateConfigName(char *new_config_name);
    void updateConfigStatus(char new_config_status);
    void addtoConfigLog(char *new_log_entry);
    void addtoConfigLog(istream&);
    void updateConfigDescription(char *, istream& );
    V_OBJECT *CONFIGURATION::updateVobjectAttachment();
    void attachVobjecttoConfig(V_OBJECT*);
    time_t setConfCreationDate();
    time_t getConfCreationDate();
    V_OBJECT *getDefaultVobject();
    ~CONFIGURATION() { Destroy(FALSE); }

```


};

```
// Description -----  
//  
// Defines a CONFIGURATION class.  
//  
// Constructor  
//  
// configuration -APL  
//  
// ONTOS required constructor  
//  
// configuration  
//  
// constructs a configuration object with the given name,  
// and manager.  
//  
// Public Members  
//  
// destroy  
//  
// Used to cleanup memory during deletion and transaction aborts.  
//  
// getDirectType  
//  
// Returns the ONTOS type for this class.  
//  
// getConfigName;  
//  
// Sends the configuration name to standard out.  
//  
// name  
//  
// Returns a pointer to the configuration name.  
//  
// getConfigStatus  
//  
// Sends the configuration status to standard out.  
//  
// getConfigManager  
//  
// Sends the manager's name for this particular configuration.  
//  
// getConfigLog  
//  
// Sends the configuration log to standard out.  
//  
// getConfigDescription  
//
```

```

// Sends the configuration description to standard output.
//
// dumpConfigSummary
//
// Provides name, version number of root vobject , date and
// description of configuration.
//
// listConfigOperators
//
// list the name of component operators in a configuration.
//
// updateConfigManager
//
// Changes the manager's name for this configuration.
//
// updateConfigName
//
// Changes the configuration name.
//
// updateConfigStatus
//
// Changes the configuration status field.
//
// addToConfigLog
//
// A log to maintain a history of the configuration.
//
// updateConfigDescription
//
// Replaces the existing description if one exist or adds a new description.
//
// attachVobjecttoConfig
//
// Adds a versioned object to configuration.
//
// setConfCreationDate
//
// Time stamp this object with the current system time.
//
// getConfCreationDate
//
// Displays the time an instance of this class was created.
//
// getDefaultVobject
//
// Returns a pointer to the attach vobject.
//
// ~configuration
//
// A destructor for the configuration class.
//

```

```
// End -----  
#endif // _CONFIGURATION_H
```

```

// File Header -----
//.....:
//.Filename.....: configuration.cxx
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char configuration_cxx_SccsId[] = "@(#)configuration.cxx 1.3\t9/16/91";

// Contents -----
//
// CONFIGURATION::CONFIGURATION ONTOS constructor
// CONFIGURATION::CONFIGURATION new instance
// CONFIGURATION::Destroy
// CONFIGURATION::getDirectType
// CONFIGURATION::getConfigName
// CONFIGURATION::name
// CONFIGURATION::getConfigStatus
// CONFIGURATION::getConfigManager
// CONFIGURATION::getConfigLog
// CONFIGURATION::getConfigDescription
// CONFIGURATION::dumpConfigSummary
// CONFIGURATION::listConfigOperators
// CONFIGURATION::updateConfigManager
// CONFIGURATION::updateConfigName
// CONFIGURATION::updateConfigStatus
// CONFIGURATION::addtoConfigurationLog - string
// CONFIGURATION::addtoConfigurationLog - file
// CONFIGURATION::updateConfigDescription
// CONFIGURATION::attachVobjecttoConfig
// CONFIGURATION::setConfCreationDate
// CONFIGURATION::getConfCreationDate
// CONFIGURATION::getDefaultVobject
// CONFIGURATION::~CONFIGURATION
//
// Description
//
// Implementation of class CONFIGURATION member functions.
//
// End -----

// Implementation Dependencies -----

```

```

#include <GlobalEntities.h>
#include <Directory.h>
#include <stream.hxx>

extern "C--"
{
#include <string.h>
}

// End Implementation Dependencies -----

// Interface Dependencies -----

#ifndef _CONFIGURATION_H
#include "configuration.h"
#endif

#ifndef _DDBDEFINES_H
#include "ddbdefines.h"
#endif

// End Interface Dependencies -----

extern Type *V_OBJECT_OType;
extern Type *CONFIGURATION_OType;

// ONTOS required constructor //

CONFIGURATION::CONFIGURATION(APL *theAPL) : (theAPL)
{
};

// New Instance Constructors //

CONFIGURATION::CONFIGURATION(char *name,
                             char *manager):(name)

// Summary -----
//
// Constructs a persistent CONFIGURATION object. This object
// contains management (header information) about a CONFIGURATION
// and a select group of modules in the configuration.
//
// Parameter
//
// name
//
// A pointer to a character string.
//
// manager
//

```

```

// A pointer to a character string.
//
// Functional Description
//
// Copies the manager's name into private data member. Initializes
// the description and log entry to null and creates a dictionary to
// hold the configuration modules.
//
// End -----
{
    initDirectType(CONFIGURATION_OType);
    config_manager = new char[strlen(manager)+1];
    strcpy(config_manager, manager);
    config_status = 'A';
    config_num_vobjects = 0;
    ConfCreationDate = setConfCreationDate();
    config_description.initToNull();
    config_log_entry.initToNull();
    theVersioned_Object.initToNull();

    putObject();
}
// End Constructor for CONFIGURATION::CONFIGURATION

// Member Functions //

void CONFIGURATION::Destroy(Boolean aborted)
{
    delete config_manager;
    if (aborted)
    {
        Object::Destroy(aborted);
    }
}

Type *CONFIGURATION::getDirectType()
{
    return CONFIGURATION_OType;
}

void CONFIGURATION::getConfigName()
{
    Directory *directory;
    char *name;

    if(!this)
    {
        cerr << "<ERROR: cannot get the name of a null CONFIGURATION>\n";
        return;
    }
}

```

```

    }
    name = Name();
    OC_getNameComponents(name, &directory, &name);
    cout << name << "\n";
}

char *CONFIGURATION::name()
{
    Directory *directory;
    char *name;

    name = Name();
    OC_getNameComponents(name, &directory, &name);
    return name;
}

void CONFIGURATION::getConfigStatus()
{
    if(!this)
    {
        cerr << "<ERROR: cannot get the Status of a null Configuration>\n";
        return;
    }
    cout << config_status << "\n";
}

void CONFIGURATION::getConfigManager()
{
    if(!this)
    {
        cerr << "<ERROR: cannot get the Manager of a null Configuration>\n";
        return;
    }
    cout << config_manager << "\n";
}

void CONFIGURATION::getConfigLog()
{
    if(!this)
    {
        cerr << "<ERROR: cannot dump the Log of a null Configuration>\n";
        return;
    }
    if(!config_log_entry)
    {
        cerr << "<Cannot display an empty log>\n";
        return;
    }
    else
    {

```

```

        TEXT_OBJECT* myTextObject = (TEXT_OBJECT*)config_log_entry.Binding(this);
        myTextObject → text(cout);
    }
}

void CONFIGURATION::getConfigDescription()
{
    if(!this)
    {
        cerr << "<ERROR: cannot get the description of a null Configuration>\n";
        return;
    }
    if(!config_description)
    {
        cerr << "<This configuration does not contain a description>\n";
        return;
    }
    else
    {
        TEXT_OBJECT* myTextObject = (TEXT_OBJECT*)config_description.Binding(this);
        myTextObject → text(cout);
    }
}

void CONFIGURATION::dumpConfigSummary()
{
    int i=0;
    cerr << "Creation Date: ";
    cout << ctime(&ConfCreationDate) << "\n";
    cerr << "Manager: ";
    getConfigManager();
    if(!theVersioned_Object)
    {
        cerr << "VOBJECT Name: ";
        cerr << "NONE ASSIGNED ";
        for (i=0;i<PRINT_VERSION_LOCATION -strlen("VOBJECT Name: NONE ASSIGNED ");i++)
            cout << " ";
        cerr << "Version Number: ";
        cerr << "NONE\n";
    }
    else
    {
        V_OBJECT *vobjectPtr = (V_OBJECT*) theVersioned_Object.Binding(this);
        cerr << "VOBJECT Name: ";
        cout << vobjectPtr → getName();
        for (i=0;i<PRINT_VERSION_LOCATION - strlen(vobjectPtr → getName());i++)
            cout << " ";
        cerr << "Version Number: ";
        cout << vobjectPtr → getVersionNumber();
    }
}

```



```

    cout << "Configuration Description follows:
\n=====\\n";
    getConfigDescription();
}

void CONFIGURATION::listConfigOperators()
{
    if(!theVersioned_Object)
    {
        cerr << "This configuration does not contain a v-object";
    }
    else
    {
        V_OBJECT *theVObjectPtr =
            (V_OBJECT*) theVersioned_Object.Binding(this);
        theVObjectPtr -> getVObjName();
        theVObjectPtr -> listOperatorNames();
    }
}

void CONFIGURATION::updateConfigManager(char *new_config_manager)
{
    if(!this)
    {
        cerr << "<ERROR: cannot change the manager of a null CONFIGURATION>\\n";
        return;
    }
    if(config_manager)
    {
        strcpy(config_manager, "");
    }
    config_manager = new char[strlen(new_config_manager)+1];
    strcpy(config_manager, new_config_manager);
    putObject();
}

void CONFIGURATION::updateConfigName(char *new_config_name)
{
    if(!this)
    {
        cerr << "<ERROR: cannot change the name of a NULL CONFIGURATION>\\n";
        return;
    }

    Name(new_config_name);
}

void CONFIGURATION::updateConfigStatus(char new_config_status)
{

```

```

    if(!this)
    {
        cerr << "<ERROR: cannot change the status of a null CONFIGURATION>\n";
        return;
    }
    config_status = new_config_status;
}

```

```

void CONFIGURATION::addtoConfigLog(char *new_log_entry)
{
    if(!config_log_entry)
    {
        TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
        textObjectPtr → append(new_log_entry);
        config_log_entry.Reset(textObjectPtr, this);
    }
    else
    {
        TEXT_OBJECT *textObjectPtr =
            (TEXT_OBJECT*) config_log_entry.Binding(this);
        textObjectPtr → append(new_log_entry);
    }
    putObject();
}

```

```

void CONFIGURATION::addtoConfigLog(istream& input_file_stream)
{
    if(!config_log_entry)
    {
        TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
        textObjectPtr → append(input_file_stream);
        config_description.Reset(textObjectPtr, this);
    }
    else
    {
        TEXT_OBJECT *textObjectPtr =
            (TEXT_OBJECT*) config_log_entry.Binding(this);
        textObjectPtr → append(input_file_stream);
    }
    putObject();
}

```

```

void CONFIGURATION::updateConfigDescription(char *fileName, istream& input_file_stream)
{
    if(!config_description)
    {
        TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
        textObjectPtr → append(fileName, input_file_stream);
    }
}

```

```

        config.description.Reset(textObjectPtr, this);
    }
    else
    {
        TEXT_OBJECT *textObjectPtr =
            (TEXT_OBJECT*) config.description.Binding(this);
        textObjectPtr → resetTheText();
        textObjectPtr → append(fileName, input_file_stream);
    }
    putObject();
}

V_OBJECT *CONFIGURATION::updateVobjectAttachment()
{
    if (!this)
    {
        cerr << "<ERROR: cannot set the v_object of a null configuration\n";
        return NULL;
    }
    V_OBJECT *vobjectPtr = getDefaultVobject();
    if (vobjectPtr)
    {
        THREAD *threadPtr = (THREAD *)OC_Lookup(vobjectPtr→getName());
        if (threadPtr)
        {
            {
                vobjectPtr = threadPtr→current();
                theVersioned_Object.Reset(vobjectPtr, this);
                putObject();
            }
        }
        return vobjectPtr;
    }
}

void CONFIGURATION::attachVobjecttoConfig(V_OBJECT *theV_Object)
{
    if (!this)
    {
        cerr << "<ERROR: cannot set the v_object of a null configuration\n";
        return;
    }
    if (!theV_Object)
    {
        cerr << "<ERROR: cannot give to a configuration a null v_object>\n";
    }
    theVersioned_Object.initToNull();
    theVersioned_Object.Reset(theV_Object, this);
}

// Member Function //

time_t CONFIGURATION::setConfCreationDate()

```

```

{
    time_t mytloc=0;
    time_t theTime;
    return theTime = time(mytloc);
}

// End -----

// Member Function //

time_t CONFIGURATION::getConfCreationDate()
{
    return ConfCreationDate;
}

// Member Function //

V_OBJECT * CONFIGURATION::getDefaultVobject()
{
    return (V_OBJECT *) (Entity *) theVersioned_Object.Binding(this);
}

// End -----

// end functions

```

```

// File Header -----
//.....:
//.Filename.....: ddbdefines.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _DDBDEFINES_H
#define _DDBDEFINES_H

    // SCCS ID follows: will compile to place date/time stamp in object file

    static char ddbdefines_h_SccsId[] = "@(#)ddbdefines.h 1.3\t9/16/91";

// Contents -----
//
// Number Defines for Evaluations functions
//
// Description
//
// These #defines are all designed just to pass information back and forth
// between the main program and the modules which evaluate the command line
// for what function to run. There are basically three types of function
// arguments CONFIGURATION arguments - beginning with a 'C', PROTOTYPE
// arguments - beginning with a 'P', and VOBJECT functions - beginning
// with a 'V'. All arguments are exactly 9 characters in length and must
// conform to one of the valid arguments in the designed interface. All
// other arguments will be invalid and return <Invalid Function> to the
// Standard I/O.
//
//
// End -----

// Interface Dependencies -----
//
// NONE
//
// End Interface Dependencies -----

#define PRINT_CONFIG_LOCATION 20
#define PRINT_VERSION_LOCATION 50
#define MAX_LINE_LENGTH 1024
#define COMMAND_TABLE_SIZE 50
#define SUCCESS TRUE
#define FAILED FALSE

```

```

#define LIST_PROTOTYPES 1
#define LONG_LIST_PROTOTYPES 81230
#define GET_PROTOTYPE_LEADER 2
#define GET_PROTOTYPE_DESCRIPTION 3
#define RETRIEVE_PROTOTYPE_DATE 5
#define INSERT_PROTOTYPE 6
#define UPDATE_PROTOTYPE_LEADER 7
#define UPDATE_PROTOTYPE_DESC 8
#define UPDATE_PROTOTYPE_NAME 9
#define GET_LATEST_CONFIGURATION 10
#define DUMP_PROTOTYPE_SUMMARY 987

#define LIST_CONFIGURATIONS 21
#define DUMP_CONFIGURATION_OPERATORS 91372
#define ADD_CONFIGURATION_OPERATORS 91378
#define LONG_LIST_CONFIGURATION_OPERATORS 92351
#define LIST_CONFIGURATION_DEFAULT_OPERATOR 6189
#define LIST_CONFIGURATION_OPERATORS 91375
#define UPDATE_CONFIGURATION_NAME 22
#define GET_CONFIGURATION_DESCRIPTION 23
#define INSERT_CONFIGURATION 24
#define UPDATE_CONFIGURATION_DESCRIPTION 25
#define GET_CONFIGURATION_MANAGER 26
#define UPDATE_CONFIGURATION_MANAGER 27
#define GET_CONFIGURATION_DATE 28
#define GET_CONFIGURATION_CHANGED 29
#define POST_CONFIGURATION_LOG 30
#define GET_CONFIGURATION_LOG 31
#define ATTACH_OPERATOR 32
#define DUMP_CONFIGURATION_SUMMARY 33
#define RELEASE_CONFIGURATION_LOCK 8124

#define LIST_OPERATORS 41
#define GET_VOBJECT_DESCRIPTION 42
#define GET_VOBJECT_DATE 43
#define GET_VOBJECT_VERSIONS 44
#define GET_VOBJECT_LOCK 45
#define GET_VOBJECT_VERSION 46
#define DUMP_VOBJECT_SUMMARY 47
#define GET_VOBJECT_POSTSCRIPT 48
#define GET_VOBJECT_GRAPH 49
#define GET_VOBJECT_IMPLEMENTATION 50
#define GET_VOBJECT_SPECIFICATION 51
#define GET_VOBJECT_SOURCE 52
#define UPDATE_VOBJECT_DESCRIPTION 53
#define ADD_VOBJECT_AND_SUBTREE 58
#define DUMP_VOBJECT_FILES 59
#define DUMP_VOBJECT_TREE 60
#define LONG_LIST_CHILDREN 61
#define LONG_LIST_PARENTS 62
#define LONG_LIST_OPERATORS 32981

```

```

#define RELEASE.SUBTREE_LOCK 8281
#define RELEASE.OPERATOR_LOCK 8992

#define ERROR.IN_EVALUATION 9999

#define LENGTH.PS_EXT 3
#define LENGTH.GRAPH_EXT 6
#define LENGTH.IMP_EXT 9
#define LENGTH.SPEC_EXT 10
#define LENGTH.SOURCE_EXT 2

#define PS_EXT ".ps"
#define GRAPH_EXT ".graph"
#define IMP_EXT ".imp.psd1"
#define SPEC_EXT ".spec.psd1"
#define SOURCE_EXT ".a"

#define DESIGN_DATABASE_DIRECTORY "^DesignDatabase"
#define PROTOTYPE_LIST "PrototypeList"
#define LONG_LIST.PROTOTYPE_UPC "PLL"
#define LONG_LIST.PROTOTYPE_LC "pll"
#define LIST.PROTOTYPE_UPC "PLN"
#define LIST.PROTOTYPE_LC "pln"
#define INSERT.PROTOTYPE_UPC "PIP"
#define INSERT.PROTOTYPE_LC "pip"
#define PROTOTYPE_EXT ".prj"

#endif // _DDBDEFINES_H

```

```

// File Header -----
//.....:
//.Filename.....: directory.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _DIRECTORY_H
#define _DIRECTORY_H

    // SCCS ID follows: will compile to place date/time stamp in
    // object file

    static char directory_h_ScscId[] = "Q(*)directory.h 1.3\t9/16/91";

// Contents -----
//
// DIRECTORY HEADER
//
// Description
//
// Defines class DIRECTORY.
//
// End -----

// Interface Dependencies -----

#ifndef _TREENODE_H
#include "treenode.h"
#endif

    // End Interface Dependencies -----

    class DIRECTORY
    {
    private:
        TREENODE_linkedlist operator_nodes;

    public:
        DIRECTORY() {};
        void read_directory(char *root_oper);
        void updatetimestamp();
        TREENODE *find_treenode(char *);
        TREENODE_linkedlist getOperatorList();
    };

```



```

// Description -----
//
// Defines class DIRECTORY. Class DIRECTORY is a non-
// persistent class.
//
// Constructor
//
// DIRECTORY
//
// Public Members
//
// read_directory
//
// Read a list of file from a directory defined by the environment
// variable PROTOTYPE, creates a corresponding list of operator nodes.
//
// updatetimestamp
//
// Updates the nodes time to reflect the time of the file
// most recently updated.
//
// find_treenode
//
// Find a given node in the list of operator nodes.
//
// getOperatorList
//
// Returns the operator node list.
//
// End Description -----
#endif // header file

```

```

// File Header -----
//.....:
//.Filename.....: directory.cxx
//.SCCS ID.....: 1.
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char directory_cxx_SccsId[] = "Q(%)directory.cxx 1.3\t9/16/91";

// Contents -----
//
// DIRECTORY::read_directory
// DIRECTORY::updatetimestamp
// DIRECTORY::find_treenode
// DIRECTORY::getOperatorList
//
// Description
//
// IMPLEMENTS class DIRECTORY CONSTRUCTORS.
//
// End -----

// Interface Dependencies -----
#include <stream.hxx>

extern "C--"
{
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>
}

#ifndef _DIRECTORY_H
#include "directory.h"
#endif

#ifndef _DDBDEFINES_H
#include "ddbdefines.h"
#endif

// ----- End Interface Dependencies -----

```

```
extern char *dirNamePtr;
```

```
void DIRECTORY::read_directory(char *root_oper)
{
```

```
    // -----
    // In body comment:
    //
    // Here I will create the list to hold those filenames that
    // contain (as a substring) the operator name. This method
    // will also scan those files that do match the pattern input
    // and throw out those with the .ps, .graph, .imp.psdl, .spec.psdl
    // and .a files. What I hope will remain is a list of only those
    // filenames which represent that operator node and it's
    // associated SUBTREE operator nodes. We'll then turn around and
    // process the resulting list into an operator tree structure and
    // compare against the database schema for storage of the
    // .ps, graph, .spec.psdl, .imp.psdl, and .a text objects into
    // the database as collected sets of COMPONENT objects.
    //
    // -----
```

```
    DIR *dirp;
    struct dirent *capsdirent;
    char *path[MAX_LINE_LENGTH];
    char *pschk = NULL;
    char *graphchk = NULL;
    char *specchk = NULL;
    char *impchk = NULL;
    char *sourcechk = NULL;
    TREENODE *operatornode = NULL;
    char *filename = NULL;
    strcpy(path, dirNamePtr);
    dirp = opendir(dirNamePtr);
    int count=0;
```

```
    TREENODE *temp;
```

```
    for (capsdirent = readdir(dirp); capsdirent != NULL;
         capsdirent = readdir(dirp))
```

```
    {
        filename=capsdirent->d_name;
```

```
        pschk = capsdirent->d_name + strlen(capsdirent->d_name) - 3;
        graphchk = capsdirent->d_name + strlen(capsdirent->d_name) - 6;
        specchk = capsdirent->d_name + strlen(capsdirent->d_name) - 10;
        impchk = capsdirent->d_name + strlen(capsdirent->d_name) - 9;
        sourcechk = capsdirent->d_name + strlen(capsdirent->d_name) - 2;
```

```
        if (strcmp(pschk, ".ps")==0)
```

```

        {
            pschk[0] = '\0';
        }
        else if (strcmp(graphchk, ".graph")==0)
        {
            graphchk[0] = '\0';
        }
        else if (strcmp(specchk, ".spec.psd1")==0)
        {
            specchk[0] = '\0';
        }
        else if (strcmp(impchk, ".imp.psd1")==0)
        {
            impchk[0] = '\0';
        }
        else if (strcmp(sourcechk, ".a")==0)
        {
            sourcechk[0] = '\0';
        }
    }

    if (strncmp(capsdirent->d_name, root_oper, strlen(root_oper))==0)
    {
        if (!(temp=find_treenode(capsdirent->d_name)))
        {
            operatornode = new TREENODE(capsdirent->d_name, NULL);
            operator_nodes.insert(operatornode);
        }
    }
}
closedir(dirp);
}

void DIRECTORY::updatetimestamp()
{
    DIR *dirp;
    struct dirent *filep;

    struct stat timestats;

    char *psfilename = NULL;
    char *graphfilename = NULL;
    char *specfilename = NULL;
    char *impfilename = NULL;
    char *sourcefilename = NULL;
    char *path[MAX_LINE_LENGTH];
    char *node_name = NULL;
    TREENODE *node;
    long temptime = 0;
    long filetime = 0;
    dirp = opendir(dirNamePtr);

```

```

slist_iterator OperatorPtr(operator_nodes);

while (node = OperatorPtr())
{
    node_name = node->getname();

    psfilename = new char [strlen(node_name)+3];
    strcpy (psfilename,node_name);
    strcat (psfilename,".ps");

    graphfilename = new char [strlen(node_name)+6];
    strcpy (graphfilename,node_name);
    strcat (graphfilename,".graph");

    impfilename = new char [strlen(node_name)+9];
    strcpy (impfilename,node_name);
    strcat (impfilename,".imp.psd1");

    specfilename = new char [strlen(node_name)+10];
    strcpy (specfilename,node_name);
    strcat (specfilename,".spec.psd1");

    sourcefilename = new char [strlen(node_name)+2];
    strcpy (sourcefilename,node_name);
    strcat (sourcefilename,".a");

    filep = readdir(dirp);
    while ((filep != NULL) && (!(strcmp(filep->d_name,psfilename)==0)))
    {
        filep = readdir(dirp);
    }
    if ((filep != NULL) && (strcmp(filep->d_name,psfilename)==0))
    {
        strcpy (path,dirNamePtr);
        strcat (path,"/");
        strcat (path,filep->d_name);
        stat(path,&timestats);
        filetime = timestats.st_ctime;
        temptime = node->get_long_time();
        node->updatetimestamp(temptime < filetime ? filetime : temptime);
    }

    rewinddir(dirp);
    filep = readdir(dirp);
    while ((filep != NULL) && (!(strcmp(filep->d_name,graphfilename)==0)))
    {
        filep = readdir(dirp);
    }
    if ((filep != NULL) && (strcmp(filep->d_name,graphfilename)==0))
    {

```

```

        strcpy (path,dirNamePtr);
        strcat (path,"/");
        strcat (path,filep→d_name);
        stat(path,&timestats);
        filetime = timestats.st_ctime;
        temptime = node→get_long_time();
        node→updatetimestamp(temptime < filetime ? filetime : temptime);
    }

    rewinddir(dirp);
    filep = readdir(dirp);
    while ((filep ≠ NULL) && (!(strcmp(filep→d_name,impfilename)==0)))
    {
        filep = readdir(dirp);
    }
    if ((filep ≠ NULL) && (strcmp(filep→d_name,impfilename)==0))
    {
        strcpy (path,dirNamePtr);
        strcat (path,"/");
        strcat (path,filep→d_name);
        stat(path,&timestats);
        filetime = timestats.st_ctime;
        temptime = node→get_long_time();
        node→updatetimestamp(temptime < filetime ? filetime : temptime);
    }

    rewinddir(dirp);
    filep = readdir(dirp);
    while ((filep ≠ NULL) && (!(strcmp(filep→d_name,specfilename)==0)))
    {
        filep = readdir(dirp);
    }
    if ((filep ≠ NULL) && (strcmp(filep→d_name,specfilename)==0))
    {
        strcpy (path,dirNamePtr);
        strcat (path,"/");
        strcat (path,filep→d_name);
        stat(path,&timestats);
        filetime = timestats.st_ctime;
        temptime = node→get_long_time();
        node→updatetimestamp(temptime < filetime ? filetime : temptime);
    }

    rewinddir(dirp);
    filep = readdir(dirp);
    while ((filep ≠ NULL) && (!(strcmp(filep→d_name,sourcefilename)==0)))
    {
        filep = readdir(dirp);
    }
    if ((filep ≠ NULL) && (strcmp(filep→d_name,sourcefilename)==0))
    {

```

```

        strcpy (path,dirNamePtr);
        strcat (path,"/");
        strcat (path,filep->d_name);
        stat(path,&timestats);
        filetime = timestats.st_ctime;
        temptime = node->get_long_time();
        node->updatetimestamp(temptime < filetime ? filetime : temptime);
    }
    rewinddir(dirp);
}
closedir(dirp);
}

```

```

TREENODE *DIRECTORY::find_treenode(char *node_name)
{
    slist_iterator list_iterator(operator_nodes);
    TREENODE *tnode;
    while (tnode=list_iterator())
        if (strcmp(tnode->getname(),node_name)==0)
            return tnode;
    return NULL;
}

```

```

TREENODE_linkedlist DIRECTORY::getOperatorList()
{
    return operator_nodes;
}

```

```

// File Header -----
//.....:
//.Filename.....: evaluation.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _EVALUATION_H
#define _EVALUATION_H

    // SCCS ID follows: will compile to place date/time stamp in object file

    static char evaluation_h_sccsid[] = "0(#)evaluation.h 1.3\t9/16/91";

// Contents -----
//
// Prototypes of functions to evaluate the command line TAG
// argument and set the appropriate case statement in main.
//
//
// End -----

// Interface Dependencies -----
//
// NONE
//
// End Interface Dependencies -----

char charupper(char c);
char *upper(char *argument);
int evaluate_configuration_function(char *function, int arguments);
int evaluate_vobject_function(char *function, int arguments);
int evaluate_prototype_function(char *function, int arguments);

// Description -----
//
// charupper
//
// Converts lower case letters to upper case.
//
// upper
//
// Converts the command line TAG field to upper case. Calls
// charupper to convert each letter.
//
// evaluate_configuration_function

```



```

//
// Determines the appropriate case statement to be executed for
// database operation pertaining to configurations.
//
// evaluate_vobject_function
//
// Determines the appropriate case statement to be executed for
// database operations pertaining to versioned objects.
//
// evaluate_prototype_function
//
// Determines the appropriate case statement to be executed for
// database operations pertaining to prototypes.
//
// End Description -----
#endif // __EVALUATION_H

```

```

// File Header -----
//.....:
//.Filename.....: evaluation.cxx
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char evaluation_cxx_SccsId[] = "@(#)evaluation.cxx 1.3\t9/16/91";

// Contents -----
//
// charupper
// upper
// evaluate_configuration_function
// evaluate_vobject_function
// evaluate_prototype_function
//
// Description
//
// Defines FUNCTIONS FOR SWITCH STMT IN MAIN.
//
// This information is required to evaluate the command
// line input and reconstruct the proper commands internal
// to the design database program.
//
//
// End -----

// Interface Dependencies -----

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <stream.h>
extern "C--"
{
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
}

#ifndef __EVALUATION_H

```

```

#include "evaluation.h"
#endif

// End Interface Dependencies -----

char charupper(char c)
{
    return islower(c) ? (c-'a'+ 'A') : c; // change char to upper case
};

char *upper(char *argument)
{
    int i;
    for (i=0; i<strlen(argument); i++) // convert argument to upper case
        argument[i] = charupper(argument[i]); // call charupper to convert each one
    return argument;
};

int evaluate_configuration_function(char *function, int arguments)
{
    if (strcmp(function, "CLN")==0)
    {
        if (arguments<1 || arguments >1)
        {
            cerr << "<ERROR: Invalid number arguments for List Configurations>\n";
            return ERROR_IN_EVALUATION;
        }
        else
            return LIST_CONFIGURATIONS; // tell main() to run List_Prototypes
    }
    else if (strcmp(function, "CUW")==0)
    {
        if (arguments<3 || arguments>3)
        {
            cerr << "<ERROR: Invalid number of arguments for Update Configuration Name>\n";
            return ERROR_IN_EVALUATION;
        }
        return UPDATE_CONFIGURATION_NAME;
    }
    else if (strcmp(function, "CGD")==0)
    {
        if (arguments<2 || arguments >2)
        {
            cerr << "<ERROR: Invalid number arguments for Get Configuration Description>\n";
            return ERROR_IN_EVALUATION;
        }
        return GET_CONFIGURATION_DESCRIPTION;
    }
    else if (strcmp(function, "CIC")==0)

```

```

    {
        if (!arguments>0)
        {
            cerr << "<ERROR: Not enough arguments for Insert Configuration>\n";
            return ERROR_IN_EVALUATION;
        }
        return INSERT_CONFIGURATION;
    }
else if (strcmp(function,"CUD")==0)
{
    if (arguments<3 || arguments>3)
    {
        cerr << "<ERROR: Invalid number arguments for Update Configuration
Description>\n";
        return ERROR_IN_EVALUATION;
    }
    return UPDATE_CONFIGURATION_DESCRIPTION;
}
else if (strcmp(function,"CGM")==0)
{
    if (arguments<2 || arguments >2)
    {
        cerr << "<ERROR: Invalid number arguments for Get Configuration Manager>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_CONFIGURATION_MANAGER;
}
else if (strcmp(function,"CDT")==0)
{
    if (arguments<3 || arguments>3)
    {
        cerr << "<ERROR: Invalid number arguments for Checkout Configuration>\n";
        return ERROR_IN_EVALUATION;
    }
    return DUMP_CONFIGURATION_OPERATORS;
}
else if (strcmp(function,"CRL")==0)
{
    if (arguments<2 || arguments>2)
    {
        cerr << "<ERROR: Invalid number arguments for Release Configuration Lock>\n";
        return ERROR_IN_EVALUATION;
    }
    return RELEASE_CONFIGURATION_LOCK;
}
else if (strcmp(function,"CAA")==0)
{
    if (arguments<2 || arguments>2)
    {
        cerr << "<ERROR: Invalid number arguments for Checkin Configuration>\n";
        return ERROR_IN_EVALUATION;
    }
}

```

```

    }
    return ADD_CONFIGURATION_OPERATORS;
}
else if (strcmp(function,"CLL")==0)
{
    if (arguments<2 || arguments>2)
    {
        cerr << "<ERROR: Invalid number arguments for List Configuration Operators>\n";
        return ERROR_IN_EVALUATION;
    }
    return LONG_LIST_CONFIGURATION_OPERATORS;
}
else if (strcmp(function,"CLV")==0)
{
    if (arguments<2 || arguments>2)
    {
        cerr << "<ERROR: Invalid number arguments for List Default Configuration
Operator>\n";
        return ERROR_IN_EVALUATION;
    }
    return LIST_CONFIGURATION_DEFAULT_OPERATOR;
}
else if (strcmp(function,"CLO")==0)
{
    if (arguments<2 || arguments>2)
    {
        cerr << "<ERROR: Invalid number arguments for List Configuration Operators>\n";
        return ERROR_IN_EVALUATION;
    }
    return LIST_CONFIGURATION_OPERATORS;
}
else if (strcmp(function,"CUM")==0)
{
    if (arguments<3 || arguments>3)
    {
        cerr << "<ERROR: Invalid number arguments for Update Configuration Manager>\n";
        return ERROR_IN_EVALUATION;
    }
    return UPDATE_CONFIGURATION_MANAGER;
}
else if (strcmp(function,"CDA")==0)
{
    if (arguments<2 || arguments >2)
    {
        cerr << "<ERROR: Invalid number of arguments for Get Configuration Date>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_CONFIGURATION_DATE;
}
else if (strcmp(function,"CDS")==0)
{

```

```

    if (arguments<2 || arguments >2)
    {
        cerr << "<ERROR: Invalid number arguments ...Dump Configuration Summary>\n";
        return ERROR_IN_EVALUATION;
    }
    return DUMP_CONFIGURATION_SUMMARY;
}
else if (strcmp(function,"CDC")==0)
{
    if (arguments<2 || arguments >2)
    {
        cerr << "<ERROR: Invalid number arguments ...Get Last Date Changed>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_CONFIGURATION_CHANGED;
}
else if (strcmp(function,"CPL")==0)
{
    if (arguments<3 || arguments>3)
    {
        cerr << "<ERROR: Invalid number arguments for Post to Configuration Log>\n";
        return ERROR_IN_EVALUATION;
    }
    return POST_CONFIGURATION_LOG;
}
else if (strcmp(function,"CGL")==0)
{
    if (arguments<2 || arguments >2)
    {
        cerr << "<ERROR: Invalid number arguments for Get Configuration Log>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_CONFIGURATION_LOG;
}
else if (strcmp(function,"CAO")==0)
{
    if (arguments<3 || arguments >4)
    {
        cerr << "<ERROR: Not enough arguments for Attach Operators>\n";
        return ERROR_IN_EVALUATION;
    }
    return ATTACH_OPERATOR;
}
else
{
    cerr << "Error in Configuration Command syntax \n\n";
}
};

```

```

int evaluate_vobject_function(char *function, int arguments)

```

```

{
    if (strcmp(function,"VLO")==0)
    {
        if (arguments<2 || arguments >3)
        {
            cerr << "<ERROR: Invalid number arguments for List Operators>\n";
            return ERROR_IN_EVALUATION;
        }
        return LIST_OPERATORS; // tell main() to run List Operators
    }
    else if (strcmp(function,"VUD")==0)
    {
        if (!arguments>0)
        {
            cerr << "<ERROR: Invalid number of arguments for Update VOBJECT Description>\n";
            return ERROR_IN_EVALUATION;
        }
        return UPDATE_VOBJECT_DESCRIPTION;
    }
    else if (strcmp(function,"VGD")==0)
    {
        if (!arguments>0)
        {
            cerr << "<ERROR: Not enough arguments to Get VOBJECT Description>\n";
            return ERROR_IN_EVALUATION;
        }
        return GET_VOBJECT_DESCRIPTION;
    }
    else if (strcmp(function,"VDD")==0)
    {
        if (!arguments>0)
        {
            cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Date>\n";
            return ERROR_IN_EVALUATION;
        }
        return GET_VOBJECT_DATE;
    }
    else if (strcmp(function,"VGV")==0)
    {
        if (arguments<2 || arguments >2)
        {
            cerr << "<ERROR: Invalid number arguments for Get VOBJECT Versions>\n";
            return ERROR_IN_EVALUATION;
        }
        return GET_VOBJECT_VERSIONS;
    }
    else if (strcmp(function,"VVV")==0)
    {
        if (arguments<1 || arguments >1)

```

```

        {
            cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Current Version>\n";
            return ERROR_IN_EVALUATION;
        }
        return GET_VOBJECT_VERSION;
    }

else if (strcmp(function,"VGL")==0)
    {
        if (arguments <2 || arguments >3)
            {
                cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Lock>\n";
                return ERROR_IN_EVALUATION;
            }
        return GET_VOBJECT_LOCK;
    }

else if (strcmp(function,"VDA")==0)
    {
        if (arguments<2 || arguments >3)
            {
                cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Summary>\n";
                return ERROR_IN_EVALUATION;
            }
        return DUMP_VOBJECT_SUMMARY;
    }

else if (strcmp(function,"VGP")==0)
    {
        if (arguments <3 || arguments >4)
            {
                cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Postscript>\n";
                return ERROR_IN_EVALUATION;
            }
        return GET_VORJECT_POSTSCRIPT;
    }

else if (strcmp(function,"VGG")==0)
    {
        if (arguments <3 || arguments >4)
            {
                cerr << "<ERROR: Invalid number of arguments for Get VOBJECT GRAPH>\n";
                return ERROR_IN_EVALUATION;
            }
        return GET_VOBJECT_GRAPH;
    }

else if (strcmp(function,"VGI")==0)
    {
        if (arguments <3 || arguments >4)
            {
                cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Implementation>\n";

```



```

        return ERROR_IN_EVALUATION;
    }
    return GET_VOBJECT_IMPLEMENTATION;
}

else if (strcmp(function,"VGC")==0)
{
    if (arguments <3 || arguments >4)
    {
        cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Specification>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_VOBJECT_SPECIFICATION;
}

else if (strcmp(function,"VGS")==0)
{
    if (arguments <3 || arguments >4)
    {
        cerr << "<ERROR: Invalid number of arguments for Get VOBJECT Source>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_VOBJECT_SOURCE;
}

else if (strcmp(function,"VDS")==0)
{
    if (arguments <2 || arguments >3)
    {
        cerr << "<ERROR: Invalid number of arguments for Dump VOBJECT Source>\n";
        return ERROR_IN_EVALUATION;
    }
    return DUMP_VOBJECT_SUMMARY;
}

else if (strcmp(function,"VAA")==0)
{
    if (arguments<2 || arguments >3)
    {
        cerr << "<ERROR: Invalid number of arguments for Add VOBJECT Subtree>\n";
        return ERROR_IN_EVALUATION;
    }
    return ADD_VOBJECT_AND_SUBTREE;
}

else if (strcmp(function,"VDF")==0)
{
    if (arguments <3 || arguments >4)
    {
        cerr << "<ERROR: Invalid number of arguments for Dump VOBJECT FILE(S)>\n";
        return ERROR_IN_EVALUATION;
    }
    return DUMP_VOBJECT_FILES;
}

else if (strcmp(function,"VLL")==0)

```

```

{
    if (!arguments>1)
    {
        cerr << "<ERROR: Invalid number of arguments for long list Operators>\n";
        return ERROR_IN_EVALUATION;
    }
    return LONG_LIST_OPERATORS;
}
else if (strcmp(function,"VLO")==0)
{
    if (!arguments>1)
    {
        cerr << "<ERROR: Invalid number of arguments for list Operators>\n";
        return ERROR_IN_EVALUATION;
    }
    return LIST_OPERATORS;
}
else if (strcmp(function,"VLP")==0)
{
    if (!arguments>1)
    {
        cerr << "<ERROR: Invalid number of arguments for list Parent/Siblings>\n";
        return ERROR_IN_EVALUATION;
    }
    return LONG_LIST_PARENTS;
}
else if (strcmp(function,"VLC")==0)
{
    if (!arguments>1)
    {
        cerr << "<ERROR: Invalid number of arguments for list Children>\n";
        return ERROR_IN_EVALUATION;
    }
    return LONG_LIST_CHILDREN;
}
else if (strcmp(function,"VDT")==0)
{
    if (!arguments>1)
    {
        cerr << "<ERROR: Invalid number of arguments for Dump VOBJECT TREE FILE(S)>\n";
        return ERROR_IN_EVALUATION;
    }
    return DUMP_VOBJECT_TREE;
}
else if (strcmp(function,"VRO")==0)
{
    if (arguments <2 || arguments >3)
    {
        cerr << "<ERROR: Invalid number of arguments for Release Operator Lock>\n";
        return ERROR_IN_EVALUATION;
    }
}

```

```

        return RELEASE_OPERATOR_LOCK;
    }
    else if (strcmp(function,"VRS")==0)
    {
        if (arguments <2 || arguments >3)
        {
            cerr << "<ERROR: Invalid number of arguments for Release Operator Subtree
Locks>\n";
            return ERROR_IN_EVALUATION;
        }
        return RELEASE_SUBTREE_LOCK;
    }

    else
    {
        cerr << "<<<Error in VOBJECT Command syntax>>> \n\n";
    }
};

```

```

int evaluate_prototype_function(char *function, int arguments)
{
    if (strcmp(function,"PLM")==0)
    {
        if (!arguments==0)
        {
            cerr << "<ERROR: Too many arguments for List Prototype Names>\n";
            return ERROR_IN_EVALUATION;
        }
        else
            return LIST_PROTOTYPES; // tell main() to run List_Prototypes
    }
    else if (strcmp(function,"PLL")==0)
    {
        if (!arguments==0)
        {
            cerr << "<ERROR: Invalid number of arguments for Long List Prototypes>\n";
            return ERROR_IN_EVALUATION;
        }
        return LONG_LIST_PROTOTYPES;
    }
    else if (strcmp(function,"PDS")==0)
    {
        if (arguments<1 || arguments >1)
        {
            cerr << "<ERROR: Invalid number of arguments for Dump Prototype Summary>\n";
            return ERROR_IN_EVALUATION;
        }
        return DUMP_PROTOTYPE_SUMMARY;
    }
}

```

```

else if (strcmp(function,"PGL")==0)
{
    if (arguments<1 || arguments >1)
    {
        cerr << "<ERROR: Invalid number of arguments for Get Prototype Leader>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_PROTOTYPE_LEADER;
}
else if (strcmp(function,"PGD")==0)
{
    if (arguments<1 || arguments >1)
    {
        cerr << "<ERROR: Invalid number arguments for Get Prototype Description>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_PROTOTYPE_DESCRIPTION;
}
else if (strcmp(function,"PRD")==0)
{
    if (arguments<1 || arguments >1)
    {
        cerr << "<ERROR: Invalid number arguments for Retrieve Prototype Date>\n";
        return ERROR_IN_EVALUATION;
    }
    return RETRIEVE_PROTOTYPE_DATE;
}
else if (strcmp(function,"PIP")==0)
{
    if (!arguments>0)
    {
        cerr << "<ERROR: Invalid number arguments for Insert Prototype>\n";
        return ERROR_IN_EVALUATION;
    }
    return INSERT_PROTOTYPE;
}
else if (strcmp(function,"PUL")==0)
{
    if (arguments<2 || arguments >2)
    {
        cerr << "<ERROR: Invalid number arguments for Update Leader>\n";
        return ERROR_IN_EVALUATION;
    }
    return UPDATE_PROTOTYPE_LEADER;
}
else if (strcmp(function,"PUD")==0)
{
    if (! arguments==1)
    {
        cerr << "<ERROR: Invalid number arguments for Update Description>\n";
        return ERROR_IN_EVALUATION;
    }
}

```

```

    }
    return UPDATE_PROTOTYPE_DESC;
}
else if (strcmp(function,"PUM")==0)
{
    if (! arguments==1)
    {
        cerr << "<ERROR: Invalid number arguments for Update Name>\n";
        return ERROR_IN_EVALUATION;
    }
    return UPDATE_PROTOTYPE_NAME;
}

else if (strcmp(function,"PGC")==0)
{
    if (arguments<1 || arguments>1)
    {
        cerr << "<ERROR: Invalid number arguments for Get_Latest_Configuration>\n";
        return ERROR_IN_EVALUATION;
    }
    return GET_LATEST_CONFIGURATION;
}
else
{
    cerr << "Error in Prototype Command syntax \n\n";
}
};

```

```

// File Header -----
//.....:
//.Filename.....: nodesupport.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _NODESUPPORT_H
#define _NODESUPPORT_H

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char nodesupport_h_SccsId[] = "@(#)nodesupport.h 1.3\t9/16/91";

// Contents -----
//
// NODESUPPPORT HEADER FILE
//
// Description
//
// SIMPLE NODE SUPPORT Functions
//
// End -----

// Interface Dependencies -----

#ifndef _TREENODE_H
#include "treenode.h"
#endif

// End Interface Dependencies -----

TREENODE* root_find(TREENODE_linkedlist list_to_search, const char* str);
int str_suffix_check(char* str, char ch);
int proper_super_string(char* str1, char* str2);
int proper_super_NODE_check(TREENODE* node_ptr, char* target_string);

// Description -----
//
// root_find
//
// Locates the root node associated with given operator name.
//
// str_suffix_check

```

```

//
// Locates the suffix of a given string.
//
// proper_super_string
//
// Determines whether one string is a prefix of another string
// (i.e., one operator is the child of another operator).
//
// proper_super_NODE_check
//
// Determines whether a given node should be added to the childlist
// of a given operator name.
//
// End Description -----
#endif // end nodesupport header file

```

```

// File Header -----
//.....:
//.Filename.....: nodesupport.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char nodesupport.cxx_ScscId[] = "c(#)nodesupport.cxx 1.3\t9/16/91";

// Contents -----
//
// root_find
// str_suffix_check
// proper_super_string
// proper_super_NODE_check
//
// End -----

// Interface Dependencies -----

extern "C--"
{
#include <string.h>
}

#ifdef _NODESUPPORT_H
#include "nodesupport.h"
#endif

// End Interface Dependencies -----

TREE_NODE* root_find(TREE_NODE linkedlist list_to_search, const char* str)

// Summary -----
//
// this function returns the address of the NODE in the list
// that has its operator_name matching *str.
//
// End Summary -----

{
    slist_iterator ret_node(list_to_search);

```



```

TREENODE* node;
while (ret_node())
{
    node = ret_node();
    if ( !strcmp(str,node→getname() ) ) return node;
}
return NULL;
}

```

```

int str_suffix_check(char* str, char ch)

```

```

    // Summary -----
    //
    // this function checks to see if the char at address str is ch
    // and that ch only appears in the string *str at this position.
    // Thus *str with ch = '.' is of the form ".example with no more
    // periods" .
    //
    // End Summary -----
{
    if (*str != ch ) return 0;
    else // check for "ch" in rest of string
    {
        if (strchr(str+1, ch ) ) return 0;
        else return 1;
    }
}

```

```

int proper_super_string(char* str1, char* str2)

```

```

    // Summary -----
    //
    // This function checks to see if str1 is a candidate to be a
    // child of str2 in the multi-way tree.
    //
    // End Summary -----
{
    // check to see if str2 is a prefix of str1
    if (str1 != strstr(str1,str2) )
        return 0;
    else
    {
        char* loc = str1 + strlen(str2);
        return str_suffix_check(loc , '.');
    }
}

```

```
int proper_super_NODE_check(TREENODE* node_ptr, char* target_string)
```

```
    // Summary -----  
    //  
    // This functions checks to see if the NODE returned by  
    // ListIterator should be added to the child_list of the NODE  
    // associated with target string. It return a 1 if it should.  
    //  
    // End Summary -----
```

```
{  
    return proper_super_string(node_ptr->getname(), target_string) ;  
}
```

```

// File Header -----
//.....:
//.Filename.....: protfunc.h
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _PROTFUNC_H
#define _PROTFUNC_H

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char protfunc_h.SccsId[] = "@(#)protfunc.h 1.3\t9/16/91";

// Contents -----
//
// Prototypes for functions related to manipulating
// instances of the class PROTOTYPE.
//
// End -----

void list_prot_func(int);
void long_list_prot_func(int);
void get_prot_leader_func(int, char*);
void get_prot_description_func(int, char*);
void dump_prot_summary_func(int, char *);
void retrieve_prot_date_func(int, char*);
void insert_prot_func(int, char*, char*, char*);
void update_prot_leader_func(int, char*, char*);
void update_prot_desc_func(int, char*, char*);
void update_prot_name_func(int, char*, char*);

// Description -----
//
// list_prot_func
//
// Provide the name of prototypes in the design database.
//
// long_list_prot_func
//
// Provides a list of all prototypes, the default configuration
// assigned to a prototype and the version number of the root
// versioned object.
//
// get_prot_leader_func

```

```

//
// Provides the name of the leader assigned to a prototype.
//
// get_prot_description_func
//
// Provides the description of a given prototype.
//
// dump_prot_summary_func
//
// Provides a summary of the prototype. Include creation date,
// leader, default configuration, and a description.
//
// retrieve_prot_date_func
//
// Provides the creation date.
//
// insert_prot_func
//
// Creates a new prototype in the database.
//
// update_prot_leader_func
//
// Changes the prototype leader's name.
//
// update_prot_desc_func
//
// Changes the description of a prototype.
//
// update_prot_name_func
//
// Changes the prototype name.
//
// End Description -----
#endif // _PROTFUNC_H

```

```

// File Header -----
//.....:
//.Filename.....: protfunc.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char protfunc_cxx_SccsId[] = "0(#)protfunc.cxx 1.3\t9/16/91";

// Contents -----
//
// list_prot_func
// long_list_prot_func
// get_prot_leader_func
// get_prot_description_func
// dump_prot_summary_func
// retrieve_prot_date_func
// insert_prot_func
// update_prot_leader_func
// update_prot_desc_func
// update_prot_name_func
//
// End -----

// Implementation Dependencies -----

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

#include <stream.hxx>
#include <List.h>
#include <Directory.h>

extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
}

#ifndef __PROTOTYPE_H
#include "prototype.h"
#endif

```

```
// End Implementation Dependencies-----
```

```
// Interface Dependencies -----
```

```
#ifndef _PROTFUNC_H  
#include "protfunc.h"  
#endif
```

```
// End Interface Dependencies -----
```

```
extern List *myPrototypeList;  
extern char *ddbRootDir;  
PROTOTYPE *prototypePtr;  
ifstream inFile;
```

```
void list_prot_func(int number_arguments)  
{  
    switch (number_arguments)  
    {  
        case 0:  
        {  
            OC_setWorkingDirectory(ddbRootDir);  
            ListIterator my_iterate(myPrototypeList);  
            while(my_iterate.moreData())  
            {  
                cout << (char *)my_iterate() << "\n";  
            }  
        }  
        break;  
        default:  
            cerr << "<ERROR: problem listing prototypes in database>\n";  
    }  
}
```

```
void long_list_prot_func(int number_arguments)  
{  
    char *proto_name = (char *)0;  
    char *configname = (char *)0;  
    switch (number_arguments)  
    {  
        case 0:  
        {  
            OC_setWorkingDirectory(ddbRootDir);  
            List &protoReference = *myPrototypeList;  
            List *proto_names = (List*)0;  
            proto_names = new List(protoReference);  
            ListIterator my_iterate(proto_names);  
            while(my_iterate.moreData())  
            {  

```

```

Directory *prototype_dir=(Directory*)0;
char *proto_name = (char *)0;
char *name = (char *)0;
proto_name = (char *)my_iterate();
name = new char [strlen(proto_name)+5];
strcpy(name,proto_name);
strcat(name,"_dir");
prototype_dir = (Directory *)OC_lookup(name);
if (prototype_dir)
    OC_setWorkingDirectory(prototype_dir);
char *prototype_name = new char [strlen(proto_name)+5];
strcpy(prototype_name,proto_name);
strcat(prototype_name,PROTOTYPE_EXT);
prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
if (prototypePtr)
{
    CONFIGURATION *configPtr;
    configPtr = prototypePtr->getDefaultConfiguration();
    if (configPtr)
    {
        configname = new char [strlen(configPtr->name()+1)];
        strcpy(configname,configPtr->name());
    }
    else
    {
        configname = new char [5];
        strcpy(configname,"NONE");
    }
    cerr << "Name: ";
    cout << proto_name;
    int i=0;
    for (i=0;i<(PRINT_CONFIG_LOCATION-strlen(proto_name));i++)
        cout << " ";
    cerr << "Default Config: ";
    cout << configname;
    for (i=0;i<(PRINT_VERSION_LOCATION-
        (PRINT_CONFIG_LOCATION+strlen(configname)));i++)
        cout << " ";
    cerr << "Version: ";
    V_OBJECT *vobjectPtr = prototypePtr->getVobject();
    if (vobjectPtr)
    {
        int version = vobjectPtr->getVersionNumber();
        cout << version << "\n";
    }
    else
        cout << "NONE \n";
    delete name;
    delete configname;
}
}

```

```

    }
    break;
default:
    cerr << "<ERROR: problem long listing prototypes in database>\n";
}
}

```

```

void get_prot_leader_func(int number_arguments, char *arg1)
{
    switch (number_arguments)
    {
        case 1:
            char *prototype_name = new char [strlen(arg1)+5];
            strcpy(prototype_name,arg1);
            strcat(prototype_name,PROTOTYPE_EXT);
            prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
            prototypePtr → getPrototypeLeader();
            break;
        default:
            cerr << "<ERROR: extra arguments in get description call\n";
    }
}

```

```

void get_prot_description_func(int number_arguments, char *arg1)
{
    switch (number_arguments)
    {
        case 1:
            char *prototype_name = new char [strlen(arg1)+5];
            strcpy(prototype_name,arg1);
            strcat(prototype_name,PROTOTYPE_EXT);
            prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
            prototypePtr → getPrototypeDescription();
            break;
        default:
            cerr << "<ERROR: extra arguments in get description call\n";
    }
}

```

```

void dump_prot_summary_func(int number_arguments, char *arg1)
{
    switch (number_arguments)
    {
        case 1:
            char *prototype_name = new char [strlen(arg1)+5];
            strcpy(prototype_name,arg1);
            strcat(prototype_name,PROTOTYPE_EXT);
            prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);

```



```

        prototypePtr → dumpPrototypeSummary();
        break;
    default:
        cerr << "<ERROR: extra arguments in dump Prototype Summary call\n";
    }
}

void retrieve_prot_date_func(int number_arguments, char *arg1)
{
    switch (number_arguments)
    {
        case 1:
            char *prototype_name = new char [strlen(arg1)+5];
            strcpy(prototype_name,arg1);
            strcat(prototype_name,PROTOTYPE_EXT);
            prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypePtr)
            {
                time_t protTime = prototypePtr → getProtCreationDate();
                cout << ctime(&protTime) << "\n";
            }
            else
            {
                cerr << "<Prototype " << arg1 << " not found>\n"
                    << "<find prototype creation date operation aborted>";
            }
            break;
        default:
            cerr << "<ERROR: invalid number args for prototype time retrieval>\n";
    }
}

```

```

void insert_prot_func(int number_arguments, char *arg1, char *arg2, char *arg3)
{
    char *prototype_name = new char [strlen(arg1)+5];
    strcpy(prototype_name,arg1);
    strcat(prototype_name,PROTOTYPE_EXT);

    prototypePtr = (PROTOTYPE *)OC_lookup(prototype_name);
    if (!prototypePtr)
    {
        switch (number_arguments)
        {
            case 1:
                prototypePtr = new PROTOTYPE(prototype_name);
                OC_setWorkingDirectory(ddbRootDir);
                myPrototypeList → Insert(arg1);
                myPrototypeList → putObject();
                break;

```

```

    case 2:
    case 3:
        prototypePtr = new PROTOTYPE(prototype_name, arg2);
        inFile.open(arg3, ios::in);
        if (inFile)
            prototypePtr → updatePrototypeDescription(arg3,inFile);
        OC_setWorkingDirectory(ddbRootDir);
        myPrototypeList → Insert(arg1);
        myPrototypeList → putObject();
        inFile.close();
        break;
    default:
        cerr << "<ERROR: invalid number args for insert prototype>\n";
    }
}
else
{
    cerr << "<NOTE: " << arg1 << " already exists!>\n";
    return;
}
}

```

```

void update_prot_leader_func(int number_arguments, char *arg1, char *arg2)
{
    switch (number_arguments)
    {
        case 2:
            char *prototype_name = new char [strlen(arg1)+5];
            strcpy(prototype_name,arg1);
            strcat(prototype_name,PROTOTYPE_EXT);
            prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
            prototypePtr → changePrototypeLeader(arg2);
            break;
        default:
            cerr << "<ERROR: invalid number args for update Leader>\n";
    }
}

```

```

void update_prot_desc_func(int number_arguments, char *arg1, char *arg2)
{
    switch (number_arguments)
    {
        case 2:
            {
                inFile.open(arg2,ios::in);
                if (!inFile)
                {
                    cerr << "File with description contents does not exist\n"
                        << "Aborting prototype updatedescription\n";
                }
            }
    }
}

```

```

    }
    else
    {
        char *prototype_name = new char [strlen(arg1)+5];
        strcpy(prototype_name,arg1);
        strcat(prototype_name,PROTOTYPE_EXT);
        prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
        prototypePtr → updatePrototypeDescription(arg2,inFile);
        inFile.close();
    }
    break;
}
default:
    cerr << "<ERROR: invalid number args for update description>\n";
}
}

```

```

void update_prot_name_func(int number_arguments, char *arg1, char *arg2)
{
    switch (number_arguments)
    {
        case 2:
            char *prototype_name = new char [strlen(arg1)+5];
            strcpy(prototype_name,arg1);
            strcat(prototype_name,PROTOTYPE_EXT);
            char *new_prototype_name = new char [strlen(arg2)+5];
            strcpy(new_prototype_name,arg2);
            strcat(new_prototype_name,PROTOTYPE_EXT);
            prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
            prototypePtr → changePrototypeName(new_prototype_name);
            break;
        default:
            cerr << "<ERROR: invalid number args for update Name>\n";
    }
}

```

```

// File Header -----
//.....:
//.Filename.....: prototype.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _PROTOTYPE_H
#define _PROTOTYPE_H

    // SCCS ID follows: will compile to place date/time stamp in
    // object file

    static char prototype_h_ScCsId[] = "0(#)prototype.h 1.3\t9/16/91";

// Contents -----
//
// PROTOTYPE
//
// Description
//
// Defines class PROTOTYPE.
//
// End -----

// Interface Dependencies -----

#include <Object.h>
#include <Dictionary.h>
#include <Reference.h>
#include "ReferenceMacros.h"

    extern "C--"
    {
#include <sys/time.h>
#include <sys/types.h>
    }

#ifndef _TEXT_OBJECT_H
#include "text_object.h"
#endif

#ifndef _CONFIGURATION_H
#include "configuration.h"
#endif

```

```
TypeCheckReference(ConfDictReference, Reference, Dictionary);
TypeCheckReference(TextObjectReference, Reference, TEXT_OBJECT);
TypeCheckReference(DefaultConfReference, Reference, CONFIGURATION);
```

```
// End Interface Dependencies -----
```

```
#define DEFAULT_NAME ""
```

```
// Class //
```

```
class PROTOTYPE : public Object
{
private:
char *protleader;
int protDictIndex;
time_t protCreationDate;
TextObjectReference protdescription;
ConfDictReference prot_configuration_list; //points to a Dictionary
DefaultConfReference prot_default_configuration;

public:
PROTOTYPE(APL *);
PROTOTYPE(char *prototype_name,
           char *prototype_leader=DEFAULT_NAME);
virtual void Destroy(Boolean aborted=FALSE);
virtual Type *getDirectType();
char *getName();
char *getConfigName();
void getPrototypeName();
void getPrototypeLeader();
void getPrototypeDescription();
void changePrototypeName(char *new_prototype_name);
void changePrototypeLeader(char *new_prototype_leader);
void updatePrototypeDescription(char*, ifstream&);
void dumpPrototypeSummary();
void addConfiguration(CONFIGURATION *configuration);
void listConfigurations();
time_t setProtCreationDate();
time_t getProtCreationDate();
void getDefaultConfigName();
CONFIGURATION *getConfiguration(char *);
CONFIGURATION *getDefaultConfiguration();
V_OBJECT *getVobject();
~PROTOTYPE() { Destroy(FALSE); }
};
```

```
// Description -----
//
// Defines a PROTOTYPE class.
```

```

//
// Constructor
//
// prototype -APL
//
// ONTOS required constructor
//
// prototype
//
// Constructs a prototype object from the given name,
// and optional team leader of the prototype.
//
// Public Members
//
// Destroy
//
// used to cleanup memory during deletion and aborts.
//
// getDirectType
//
// Return the ONTOS Type of class PROTOTYPE.
//
// getName
//
// Returns a character string containing the name of the prototype
//
// getConfigName
//
// Returns a character string containing the last configuration
// worked on by a user
//
// getPrototypeName
//
// Prints the prototype name to standard output
//
// getPrototypeLeader
//
// Displays the prototype designer team leader's name.
//
// getPrototypeProtdescription
//
// Displays a protdescription of the prototype.
//
// changePrototypeName
//
// Change the name of the prototype.
//
// changePrototypeLeader
//
// Change the prototype leader's name.
//

```

```

// updatePrototypeProtdescription
//
// Adds a protdescription to a PROTOTYPE object.
//
// dumpPrototypeSummary
//
// Provides date created, leader, default configuration,
// and description of a prototype.
//
// addConfiguration
//
// Adds a configuration with a given name to the prototype
//
// listConfigurations
//
// List the names of all the configurations in the prototype.
//
// getDefaultConfigName
//
// Displays the name of the default configuration.
//
// getConfiguration
//
// Used by this class as a support function to update the default
// configuration.
//
// getDefaultConfiguration
//
// returns a pointer to the default Configuration
//
// getVobject
//
// returns the most current root V_OBJECT (Operator or Type)
// associated with a prototype.
//
// ~prototype
//
// The class destructor.
//
// End -----
#endif // _PROTOTYPE_H

```

```

// File Header -----
//.....:
//.Filename.....: prototype.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char prototype_cxx_ScCsId[] = "@(#)prototype.cxx 1.3\t9/16/91";

// Contents -----
//
// PROTOTYPE::PROTOTYPE ONTOS constructor
// PROTOTYPE::PROTOTYPE constructor
// PROTOTYPE::Destroy
// PROTOTYPE::getDirectType
// PROTOTYPE::getName
// PROTOTYPE::getConfigName
// PROTOTYPE::getPrototypeName
// PROTOTYPE::getPrototypeLeader
// PROTOTYPE::getPrototypeDescription
// PROTOTYPE::changePrototypeName
// PROTOTYPE::changePrototypeLeader
// PROTOTYPE::updatePrototypeDescription
// PROTOTYPE::dumpPrototypeSummary
// PROTOTYPE::addConfiguration
// PROTOTYPE::listConfigurations
// PROTOTYPE::setProtCreationDate
// PROTOTYPE::getProtCreationDate
// PROTOTYPE::getDefaultConfigName
// PROTOTYPE::getConfiguration
// PROTOTYPE::getDefaultConfiguration
// PROTOTYPE::getVobject
//
// Description
//
// Implementation of class PROTOTYPE member functions.
//
// End -----

// Interface Dependencies -----

#include <GlobalEntities.h>
#include <Directory.h>

```



```

#include <stream.hxx>

extern "C--"
{
#include <sys/time.h>
#include <strings.h>
}

#ifndef _PROTOTYPE_H
#include "prototype.h"
#endif

// End Interface Dependencies -----

extern Type *PROTOTYPE_OType;
extern Type *CONFIGURATION_OType;

// ONTOS required constructor //

PROTOTYPE::PROTOTYPE(APL *theAPL) : (theAPL)
{
};

// New Instance Constructor //

PROTOTYPE::PROTOTYPE(char *prototype_name,
                     char *prototype_leader) : (prototype_name)

// Summary -----
//
// Constructs a persistent prototype object. A PROTOTYPE object
// contains general management information about a prototype and
// a reference to a container holding configuration objects.
//
// Parameter
//
// prototype_name
//
// A pointer to a character string.
//
// prototype_leader
//
// A pointer to a character string.
//
// Functional Description
//
// Passes the object name to class Object. Copies the leader's name
// into private data members. We initialize the description to null
// and create a dictionary to hold various configurations.
//

```

```

// End -----

{
    initDirectType(PROTOTYPE_OType);

    protleader = new char[strlen(prototype_leader)+1];
    strcpy(protleader, prototype_leader);
    protDictIndex = 0;
    protCreationDate = setProtCreationDate();

    protdescription.initToNull();
    prot_default_configuration.initToNull();

    if(!CONFIGURATION_OType)
    {
        CONFIGURATION_OType = (Type*)OC_lookup("CONFIGURATION");
    }

    Dictionary *new_configuration = new Dictionary(OC_integer,
                                                    CONFIGURATION_OType,
                                                    TRUE, FALSE);

    new_configuration →putObject();
    prot_configuration_list.Reset(new_configuration, this);

    putObject();
}

// End Constructor PROTOTYPE::PROTOTYPE //

// Member Function //

void PROTOTYPE::Destroy(Boolean aborted)
{
    delete protleader;
    if (aborted)
    {
        Object::Destroy(aborted);
    }
}

Type* PROTOTYPE::getDirectType()

// Summary -----
//
// returns the ONTOS Type for the prototype class.
//
// Return value
//
// A pointer to an ONTOS Type.
//
// End -----

```

```

{
    return PROTOTYPE_OType;
}

```

```

char * PROTOTYPE::getName()

```

```

    // Summary -----
    //
    // returns the name of the prototype
    //
    // Return value
    //
    // A pointer to a character string
    //
    // End -----

```

```

{
    Directory *directory=(Directory *)0;
    char *name=(char *)0;
    char *temp=(char *)0;

    if(!this)
    {
        cerr << "<ERROR: cannot get the name of a null PROTOTYPE>\n";
        return NULL;
    }
    else
    {
        name = Name();
        OC_getNameComponents(name, &directory, &name);
        temp = new char [strlen(name)+1];
        temp = strtok(name, ".");
        return temp;
    }
}

```

```

void PROTOTYPE::getPrototypeName()

```

```

    // Summary -----
    //
    // Displays the name of the prototype
    //
    // Return value
    //
    // Displays the Prototype Name and a linefeed to the stdout
    //
    // End -----

```

```

{
    Directory *directory;

```

```

char *name;

if(!this)
{
    cerr << "<ERROR: cannot get the name of a null PROTOTYPE>\n";
    return;
}
else
{
    name = Name();
    OC_getNameComponents(name, &directory, &name);
    cout << name << "\n";
}
}

void PROTOTYPE::getPrototypeLeader()

    // Summary -----
    //
    // Displays the name of the prototype leader
    //
    // Return value
    //
    // Displays the Prototype leader and a linefeed to the stdout
    //
    // End -----

{
    if(!this)
    {
        cerr << "<ERROR: cannot get the leader's name of a null PROTOTYPE>\n";
        return;
    }
    cout << protleader << "\n";
}

void PROTOTYPE::getPrototypeDescription()

    // Summary -----
    //
    // Displays the prototype description
    //
    // Return value
    //
    // Displays description to stdout if a description exists
    //
    // End -----

{
    if(!this)

```

```

    {
        cerr << "<ERROR: cannot get the description of a null PROTOTYPE>\n";
        return;
    }
    if(!protdescription)
    {
        cerr << "<This prototype does not contain a description>\n";
        return;
    }
    else
    {
        TEXT_OBJECT* myTextObjPtr = (TEXT_OBJECT*) protdescription.Binding(this);
        myTextObjPtr -> text(cout);
    }
}

```

```

void PROTOTYPE::changePrototypeName(char *new_prototype_name)

```

```

    // Summary -----
    //
    // Changes the prototype name
    //
    // Parameter
    //
    // new_prototype_name
    //
    // a character string pointer containing the new name
    //
    // Return value
    //
    // N/A
    //
    // End -----

```

```

{
    if(!this)
    {
        cerr << "<ERROR: cannot set the name of a null PROTOTYPE>\n";
        return;
    }
    Name(new_prototype_name);
}

```

```

void PROTOTYPE::changePrototypeLeader(char *new_prototype_leader)

```

```

    // Summary -----
    //
    // changes the prototype leader
    //
    // Parameter

```

```

//
// new_prototype_leader
//
// a character string pointer containing the new leader's name
//
// Return value
//
// N/A
//
// End -----

{
    if(!this)
    {
        cerr << "<ERROR: cannot change the leader of a null PROTOTYPE>\n";
        return;
    }
    delete protleader;
    protleader = new char[strlen(new_prototype_leader)+1];
    strcpy(protleader, new_prototype_leader);
    putObject();
}

void PROTOTYPE::updatePrototypeDescription(char *fileName, ifstream& input_file_stream)

// Summary -----
//
// changes the prototype description
//
// Parameter
//
// fileName
//
// a character string pointer containing the new name of
// the file containing the new description.
//
// input_file_stream
//
// the file handle of the input description file.
//
// Return value
//
// N/A
//
// End -----

{
    if(!protdescription)
    {
        TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
        textObjectPtr -> append(fileName, input_file_stream);
    }
}

```

```

        protdescription.Reset(textObjectPtr, this);
        putObject();
    }
    else
    {
        TEXT_OBJECT *textObjectPtr = (TEXT_OBJECT*) protdescription.Binding(this);
        textObjectPtr → resetTheText();
        textObjectPtr → append(fileName, input_file_stream);
        putObject();
    }
}

```

```

void PROTOTYPE::dumpPrototypeSummary()
{

```

```

    // Summary -----
    //
    // Displays the date created, leader, default config, and
    // description of a prototype to stdout
    // 1 item per line ending with the (potentially) multi-line
    // description.
    //
    // Parameter
    //
    // N/A
    //
    // Return value
    //
    // N/A
    //
    // End -----

    time_t creationdate = 0;
    creationdate = getProtCreationDate();
    cerr << "Creation Date: ";
    cout << ctime(&creationdate) << "\n";
    cerr << "Leader: ";
    getPrototypeLeader();
    cerr << "Default Config: ";
    getDefaultConfigName();
    cerr << "Prototype Description
follows:\n===== \n\n";
    getPrototypeDescription();
}

```

```

void PROTOTYPE::addConfiguration(CONFIGURATION *configuration)
{

```

```

    // Summary -----
    //
    // adds a configuration to the prototype

```

```

//
// Parameter
//
// configuration
//
// a pointer to the configuration to be added to the
// prototype.
//
//
// Return value
//
// N/A
//
// End -----

if(!this)
{
    cerr << "<ERROR: cannot set the description of a null PROTOTYPE>\n";
    return;
}
if(!configuration)
{
    cerr << "<ERROR: cannot give to a PROTOTYPE a null configuration>\n";
    return;
}
else
{
    protDictIndex = protDictIndex + 1;
    Dictionary*confDictionaryPtr=
        (Dictionary*)prot_configuration_list.Binding(this);
    confDictionaryPtr → Insert(protDictIndex, (Entity *)configuration);
    confDictionaryPtr → putObject();
    prot_default_configuration.Reset(configuration, this);
    putObject();
}
}

void PROTOTYPE::listConfigurations()

// Summary -----
//
// Display the configuration names contained in this prototype
// to stdout. 1 configuration / line.
//
// Parameter
//
// N/A
//
// Return value
//
// N/A

```



```

//
// End -----

{
    CONFIGURATION *the_configuration;
    char *name;
    Directory *directory;

    if(!prot_configuration_list)
    {
        cerr << "<ERROR: cannot return a prototype from an empty list>\n";
        return;
    }

    Dictionary *confDictionaryPtr =
        (Dictionary*)prot_configuration_list.Binding(this);
    DictionaryIterator configlist_iterator(confDictionaryPtr);

    while(the_configuration =(CONFIGURATION*)(Entity*)configlist_iterator())
    {
        if (name = the_configuration->Name());
        {
            OC_getNameComponents(name, &directory, &name);
            cout << name << "\n";
        }
    }
}

//Member Function //

time_t PROTOTYPE::setProtCreationDate()

// Summary -----
//
// sets the creation date to system date at time of this call.
//
// Parameter
//
// N/A
//
// Return value
//
// time_t as a default long value containing the system time
// This function as a byproduct updates the protCreationDate
// attribute field.
//
// End -----

{
    time_t mytloc=0;

```

```

    time_t theTime;
    return theTime = time(mytloc);
}

// End -----

// Member Function //

time_t PROTOTYPE::getProtCreationDate()

    // Summary -----
    //
    // Returns the prototype's creation date
    //
    // Parameter
    //
    // N/A
    //
    // Return value
    //
    // time_t as a long value containing the system time
    //
    // End -----

{
    return protCreationDate;
}

// End -----

void PROTOTYPE::getDefaultConfigName()

    // Summary -----
    //
    // Displays the default Configuration name for this prototype
    // to stdout
    //
    // Parameter
    //
    // N/A
    //
    // Return value
    //
    // N/A
    //
    // End -----

{
    Directory *directory;

```

```

char *name;

if(!prot_default_configuration)
{
    cerr << "\n\n<No configurations are contained in this prototype.>\n\n";
}
else
{
    CONFIGURATION *the_configuration =
        (CONFIGURATION*)prot_default_configuration.Binding(this);
    name = the_configuration -> Name();
    OC_getNameComponents(name, &directory, &name);
    cout << name << "\n";
}
}

char * PROTOTYPE::getConfigName()
{
    // Summary -----
    //
    // Returns a character string pointer to an area in memory
    // containing the name of the default Configuration name
    // for this prototype.
    //
    // Parameter
    //
    // N/A
    //
    // Return value
    //
    // character string pointer
    //
    // End -----

    Directory *directory;
    char *name;

    if(!prot_default_configuration)
    {
        cerr << "<No configurations are contained in this prototype.>\n\n";
        return NULL;
    }
    else
    {
        CONFIGURATION *the_configuration =
            (CONFIGURATION*)prot_default_configuration.Binding(this);
        name = the_configuration -> Name();
        OC_getNameComponents(name, &directory, &name);
        return name;
    }
}

```

```
}
```

```
CONFIGURATION *PROTOTYPE::getConfiguration(char *confName)
```

```
// Summary -----  
//  
// Used by this class as a support function to update the default  
// configuration.  
//  
// Parameter  
//  
// confName  
//  
// configuration name to lookup the default configuration  
// for this prototype  
//  
// Return value  
//  
// Configuration pointer if successful. Null pointer if  
// failed.  
//  
// End -----
```

```
{  
    CONFIGURATION *myConfPtr = (CONFIGURATION *)OC_lookup(confName);  
    if (myConfPtr)  
    {  
        return myConfPtr;  
    }  
    else  
    {  
        return (CONFIGURATION*)0;  
    }  
}
```

```
CONFIGURATION *PROTOTYPE::getDefaultConfiguration()
```

```
// Summary -----  
//  
// Returns the default configuration for this prototype.  
// The last configuration worked on.  
//  
// Parameter  
//  
// N/A  
//  
// Return value  
//  
// Configuration pointer if successful. Null pointer if  
// failed.  
//
```

```

        // End -----
    {
        CONFIGURATION *the_configuration =(CONFIGURATION *)0;

        if(!prot_default.configuration)
        {
            return NULL;
        }
        else
        {
            the_configuration =
                (CONFIGURATION*)prot_default.configuration.Binding(this); return the_configuration;
        }
    }
}

```

V_OBJECT *PROTOTYPE::getVobject()

```

    // Summary -----
    //
    // This method assumes that the root versioned object (V_OBJECT)
    // has the same name as the prototype. If this is not the case
    // then this function will not work. Taking the prototype name,
    // a thread lookup is performed and the most current V_OBJECT
    // in the thread is returned.
    //
    // Parameter
    //
    // N/A
    //
    // Return value
    //
    // V_OBJECT pointer if successful. Null pointer if
    // failed.
    //
    // End -----

```

```

{
    THREAD *threadPtr=(THREAD*)0;
    char *name = new char [strlen(getName())+1];
    strcpy(name,getName());
    threadPtr = (THREAD *)OC_Lookup(name);
    if (threadPtr)
    {
        V_OBJECT *vobjectPtr = threadPtr->current();
        if (vobjectPtr)
            return vobjectPtr;
        else
        {
            return NULL;
        }
    }
}

```

```
    }  
    else  
    {  
        return NULL;  
    }  
}
```

```
// end functions
```

```

// File Header -----
//.....:
//.Filename.....: queue.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _QUEUE_H
#define _QUEUE_H

    // SCCS ID follows: will compile to place date/time stamp in
    // object file

    static char QUEUE_h_ScscId[] = "Q(%)queue.h 1.3\t9/16/91";

// Contents -----
//
// QUEUE
//
// Description
//
// IMPLEMENTS class QUEUE CONSTRUCTORS.
//
// End -----

// Interface Dependencies -----
//

#include <iostream.h>

    extern "C--" {
#include <stdio.h>
#include <string.h>
    }

//
// End Interface Dependencies -----

// Description -----
//
// Defines the slink, slist, slistIterator,
// treenode_linkedlist, and treenode_queue classes
//
//
// There are no ontos mechanisms in these classes. The primary
// purpose of these classes is to provide the support structures

```

```

// of linked lists and queues in order to process the CAPS
// PROTOTYPE subdirectory.
//
// As a subdirectory is read, each file is analyzed to determine
// whether it is an operator/type. If found to belong to
// an operator which might version, the operator and operator
// information is placed into a multi-way tree paralleling the
// decomposition of an operator in the CAPS system. The
// TREENODES are then compared against operator structures in
// the Ontos Database in other programs documented elsewhere.
//
// These structures are simple in nature and can be found in
// any good C++ textbook. These particular examples came from
// Bjarne Stroustrup's C++ Programming Language textbook (pg 203).
// Please refer to the textbook for further explanation of the
// data structures and how they are manipulated.
//
// End -----

```

```

class TREENODE;
class slist;
class slist_iterator;

```

```

class slink {
    friend class slist;
    friend class slist_iterator;

private:
    slink* next;
    TREENODE * e;
    slink(TREENODE * a, slink* p);
};

```

```

class slist {
    friend class slist_iterator;

```

```

private:
    slink* last; // last-> next is head of list

```

```

public:
    slist(); // { last = NULL; }
    slist(TREENODE * a);
    int insert(TREENODE * a); // add at head of list
    int append(TREENODE * a); // add at tail of list
    TREENODE * get(); // return and remove head of list
    void clear(); // remove all links
    int empty(); // returns 1 if list is empty
    ~slist(); // { clear(); }
};

```



```

class slist_iterator{
private:
    slist* ce;
    slist* cs;

public:
    slist_iterator(slist& s);
    TREENODE * operator()();
};

class TREENODE_linkedlist : public slist
{
public:
    TREENODE_linkedlist();
    int member(char *name);
};

class TREENODE_queue : private slist
{
public:
    TREENODE_queue(){}
    void put(TREENODE * a) {append(a); }
    slist::empty;
    slist::get;
};

#endif // QUEUE Class header

```

```

// File Header -----
//.....:
//.Filename.....: queue.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char QUEUE_cxx_SccsId[] = "@(#)queue.cxx 1.3\t9/16/91";

// Contents -----
//
// QUEUE
//
// Description
//
// IMPLEMENTS class QUEUE CONSTRUCTORS.
//
// End -----

// Interface Dependencies -----
//

#ifndef _QUEUE_H
#include "queue.h"
#endif
//
// End Interface Dependencies -----

//-----X-----X-----X-----X-----X
// these are the implementation methods of classes
// slink, slist, slist_iterator, and TREENODE_linkedlist
//-----X-----X-----X-----X-----X

//-----X-----X-----X-----X-----X
//
// slink methods
//
//-----X-----X-----X-----X-----X

slink::slink(TREENODE * a, slink* p)

{
    e = a;

```

```

    next = p;
}

//-----X-----X-----X-----X-----X
//
// slist methods
//
//-----X-----X-----X-----X-----X

slist::slist()
{
    last = NULL;
}

slist::slist(TREENODE * a)
{
    last = new slink(a, NULL);
    last → next = last;
}

int slist::insert(TREENODE * a)
{
    if (last)
        last → next = new slink(a, last → next);
    else {
        last = new slink(a, NULL);
        last → next = last;
    }
    return 0;
}

int slist::append(TREENODE * a)
{
    if (last)
        last = last → next = new slink(a, last → next);
    else
    {
        last = new slink(a, NULL);
        last → next = last;
    }
    return 0;
}

TREENODE * slist::get()
{
    // improve the following line for better error detection
    if (last == NULL) cout << "get from empty slist\n";
    slink* f = last → next;
    TREENODE * r = f → e;

```

```

    if (f == last) last = NULL;
    else last → next = f → next;
    delete f;
    return r;
}

```

```

void slist::clear()
{
    slink* l = last;
    if ( l == NULL ) return;
    do
    {
        slink* ll = l;
        l = l → next;
        delete ll;
    } while (l ≠ last);
}

```

```

int slist::empty()
{
    if (last == NULL) return 1;
    else return 0;
}

```

```

slist::~slist()
{
    clear();
}

```

```

//-----X-----X-----X-----X-----X
//
// slist iterator methods
//
//-----X-----X-----X-----X-----X

```

```

slist_iterator::slist_iterator(slist& s)

```

```

{
    cs = &s;
    ce = cs → last;
}

```

```

TREENODE *slist_iterator::operator()()
{
    TREENODE * ret = ce ? (ce = ce → next) → e : NULL;
    if ( ce == cs → last) ce = NULL;
    return ret;
}

```

```
//-----X-----X-----X-----X-----X
//
// TREENODE_linkedlist methods
//
//-----X-----X-----X-----X-----X
```

```
TREENODE linkedlist::TREENODE linkedlist()
{
}
```

```
int TREENODE_linkedlist::member(char *name)
```

```
{
    char *temp = name;
    return 1;
}
```

```

// File Header -----
//.....:
//.Filename.....: text_object.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _TEXT_OBJECT_H
#define _TEXT_OBJECT_H

    // SCCS ID follows: will compile to place date/time stamp in object file

    static char text_object_h_SccsId[] = "@(#)text_object.h 1.3\t9/16/91";

// Contents -----
//
// TEXT_OBJECT
//
// Description
//
// Defines class TEXT_OBJECT.
//
// End -----

// Interface Dependencies -----
// SCCS ID follows: will compile to place date/time stamp in object file

// Interface Dependencies -----

#include <Object.h>
#include <stream.hxx>

// End Interface Dependencies -----

class TEXT_OBJECT : public Object
{
private:
    char *the_file_name;
    char *the_text;
public:
    TEXT_OBJECT(APL *);
    TEXT_OBJECT();
    void Destroy(Boolean aborted=FALSE);
    Type *getDirectType();

```

```

void append(char *, ifstream&);
void append(char *);
void append(ifstream&);
void text(ostream&); // standard output
    Boolean rebuildTextFile(char*);
void displayFileName();
char *getFileName();
char *text();
void resetTheText();
~TEXT_OBJECT() { Destroy(FALSE); }
};

// Description -----
//
// Defines a TEXT_OBJECT class. The class TEXT_OBJECT is a derived
// class of Object.
//
// Constructor
//
// TEXT_OBJECT - APL
//
// ONTOS required constructor
//
// TEXT_OBJECT
//
// Creates a new instance of TEXT_OBJECT
//
// Public Members
//
// Destroy
//
// ONTOS required method.
//
// getDirectType
//
// ONTOS required method used to return the class Type.
//
// append
//
// Reads in a text file.
//
// append
//
// Read in a string append to existing string in log fashion.
//
// text
//
// Send the text contents to standard output.
//
// rebuildTextFile
//

```

```

// Write the contents of a text object to a file.
//
// displayFileName
//
// Send the file name to standard output.
//
// getFileName
//
// Return a pointer the file name contained in the text object.
//
// text
//
// Returns a pointer to the text contained in the text object.
//
// resetTheText
//
// Set the text field to a empty string.
//
// ~TEXT_OBJECT
//
// class destructor.
//
// End Description -----
#endif // _TEXT_OBJECT_H

```



```

// File Header -----
//.....:
//.Filename.....: text_object.cxx
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp
// in object file

static char composite_cxx_ScscId[] = "0(0)text_object.cxx 1.3\t9/16/91";

// Contents -----
//
// TEXT_OBJECT::TEXT_OBJECT Ontos Constructor
// TEXT_OBJECT::TEXT_OBJECT
// TEXT_OBJECT::Destroy
// TEXT_OBJECT::getDirectType
// TEXT_OBJECT::append
// TEXT_OBJECT::append
// TEXT_OBJECT::append
// TEXT_OBJECT::text
// TEXT_OBJECT::rebuildTextFile
// TEXT_OBJECT::displayFileName
// TEXT_OBJECT::getFileName
// TEXT_OBJECT::text
// TEXT_OBJECT::resetTheText
//
// Description
//
// Implementation of class TEXT_OBJECT member functions.
//
// End -----

// Interface Dependencies -----

#include <strstream.h>

#ifndef _TEXT_OBJECT_H
#include "text_object.h"
#endif

#ifndef _TRACER_H
#include "tracer.h"
#endif

```

```

#ifdef _DDBDEFINES_H
#include "ddbdefines.h"
#endif

extern "C--"
{
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <stdlib.h>
#include <strings.h>
}
// End -----

extern Type *TEXT_OBJECT_Type;
extern char *dirNamePtr;

TEXT_OBJECT::TEXT_OBJECT(APL *theAPL) : Object(theAPL)

    // Summary -----
    //
    // Ontos Required Constructor
    //
    // Return value
    //
    // A TEXT_OBJECT object
    //
    // End -----

{
};

TEXT_OBJECT::TEXT_OBJECT()

    // Summary -----
    //
    // Constructor
    //
    // Return value
    //
    // A TEXT_OBJECT object
    //
    // the_file_name and the_text attributes are initialized to NULL;
    // End -----

{
    the_text=new char[1];
    strcpy(the_text,"");
    the_file_name = new char[1];
    strcpy(the_file_name, "");
}

```

```

    putObject();
};

void TEXT_OBJECT::Destroy(Boolean aborted)

    // Summary -----
    //
    // Constructor
    //
    // Return value
    //
    // A TEXT_OBJECT object
    //
    // the_file_name and the_text attributes are initialized to NULL;
    // End -----
{
    if(the_file_name);
    delete the_file_name;
    delete the_text;
    if(aborted)
    {
        Object::Destroy(aborted);
    }
};

```

```

Type *TEXT_OBJECT::getDirectType()

```

```

    // Summary -----
    //
    // Ontos required method which returns the type of this object.
    //
    // Parameter
    //
    // N/A
    //
    // Return Value
    //
    // the type of this object. TEXT_OBJECT.
    //
    // End -----
{
    return TEXT_OBJECT.OType;
};

```

```

void TEXT_OBJECT::append(char *filename, ifstream& input_file)

```

```

    // Summary -----
    //
    // append a file as a text object
    //
    // Parameter

```

```

//
// filename
//
// character string * containing the name of the file which
// is read into a text_object.
//
// input_file
//
// file handle of input file
//
// Return Value
//
// stores the file as a text_object in the database
// End -----

{
    the_file_name = new char[strlen(filename) + 1];
    strcpy(the_file_name, filename);

    ostringstream buf;
    char ch;
    while (buf && input_file.get(ch))
    {
        buf.put(ch);
    }
    the_text = buf.str();
    putObject();
};

void TEXT_OBJECT::append(char *instring)

// Summary -----
//
// append a character string as a text object. stores the character
// string as a text_object in the database
//
// Parameter
//
// instring
//
// character string * containing the description to be added as
// a text_object.
//
// Return Value
//
// N/A
//
// End -----

{
    time_t mytloc=0;

```

```

time_t theTime;

char *temp_text=the_text;
the_text=new char[strlen(temp_text)+strlen(instring)+
                  strlen(ctime(&theTime))+1];

strcpy(the_text,temp_text);
strcat(the_text, "\n");

theTime = time(mytloc);
strcat(the_text, ctime(&theTime));
strcat(the_text, "\n");

strcat(the_text,instring);
strcat(the_text, "\n");

putObject();
};

void TEXT_OBJECT::append(ifstream& input_file)

    // Summary -----
    //
    // append a file as a text object. stores the file as a
    // text_object in the database
    //
    // Parameter
    //
    // input_file
    //
    // file handle of input file
    //
    // Return Value
    //
    // N/A
    //
    // End -----
{
    time_t mytloc=0;
    time_t theTime = 0;

    char *temp_text=the_text; //save old text

    ostringstream buf;
    char ch;
    while (buf && input_file.get(ch))
    {
        buf.put(ch);
    }

    theTime = time(mytloc);

```

```

char *file_text = buf.str();
the_text=new char[strlen(temp_text)+strlen(file_text) +
                  strlen(ctime(&theTime))+256];
int tempsize = strlen(temp_text)+strlen(file_text) +
               strlen(ctime(&theTime))+256 ;
strcpy(the_text,temp_text);
strcat(the_text, "\n");
strcat(the_text, ctime(&theTime));
strcat(the_text, "\n");
strcat(the_text,file_text);
strcat(the_text, "\n");

putObject();
}

```

```

void TEXT_OBJECT::text(ostream& outstream)

```

```

    // Summary -----
    //
    // output the text_object as a file. dumps the text_object to
    // the PROTOTYPE environment subdirectory
    //
    // Parameter
    //
    // outstream
    //
    // file handle of output file
    //
    // Return Value
    //
    // N/A
    //
    // End -----

{
    outstream << the_text;
};

```

```

Boolean TEXT_OBJECT::rebuildTextFile(char *fileMode)

```

```

    // Summary -----
    //
    // output the text_object as a file in "r" - read only or "w" read
    // and u - write mode (refer to Unix system manual). dumps the
    // text_object to the PROTOTYPE environment subdirectory
    //
    // Parameter
    //
    // fileMode
    //

```

```

// "r" - read only. "w" read/write.
//
// Return Value
//
// Boolean SUCCESS or FAILURE - refers to success of
// rebuilding file on the disk.
//
// End -----
{
    ofstream oFile;
    char *mypath = new char[MAX_LINE_LENGTH];
    strcpy(mypath, dirNamePtr);
    strcat(mypath, "/");
    strcat(mypath, the_file_name);
    if (strcmp(fileMode, "w") == 0 || strcmp(fileMode, "W") == 0)
    {
        oFile.open(mypath, ios::noreplace);
        if (!oFile)
        {
            return FAILED;
        }
    }
    else
    {
        oFile.open(mypath, ios::noreplace, 0444);
        if (!oFile)
        {
            cerr << "<ERROR: file " << the_file_name << " already exists.>\n";
            return FAILED;
        }
    }
    if (the_text)
        oFile << the_text ;
    oFile.close();
    return SUCCESS;
}

void TEXT_OBJECT::displayFileName()
// Summary -----
//
// Displays the text_object filename to stdout.
//
// Parameter
//
// N/A
//
// Return Value
//
// N/A
//
// End -----

```

```

{
    cout << the_file_name << "\n";
}

char *TEXT_OBJECT::getFileName()
    // Summary -----
    //
    // Returns the attribute containing the name of the file
    // as it was stored on the disk.
    //
    // Parameter
    //
    // N/A
    //
    // Return Value
    //
    // character string containing the file name of the object
    //
    // End -----
{
    return the_file_name;
}

```

```

char *TEXT_OBJECT::text()

    // Summary -----
    //
    // return the contents of the_text
    //
    // Parameter
    //
    // N/A
    //
    // Return Value
    //
    // character string pointer with the text in the text_object.
    //
    // End -----
{
    return the_text;
};

```

```

void TEXT_OBJECT::resetTheText()
    // Summary -----
    //
    // Reinitialize the_text attribute to a blank character.
    //
    // Parameter
    //
    // N/A

```



```
//  
// Return Value  
//  
// N/A  
//  
// End -----  
{  
    strcpy(the_text, "");  
}
```

```
// File Header -----
//.....:
//.Filename.....: thread.h
//.SCCS ID.....: 1.3
//.Release No.....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----
```

```
#ifndef _THREAD_H
#define _THREAD_H
```

```
// SCCS ID follows: will compile to place date/time stamp in
// object file
```

```
static char thread_h.SccsId[] = "0(#)thread.h 1.3\t9/16/91";
```

```
// Contents -----
//
// THREAD
//
// Description
//
// Defines class THREAD.
//
// End -----
```

```
// Interface Dependencies-----
```

```
#include <Object.h>
#include <Dictionary.h>
#include <Reference.h>
#include "ReferenceMacros.h"
#include <stream.hxx>
```

```
class V_OBJECT;
```

```
// End Interface Dependencies -----
```

```
TypeCheckReference(VOListReference, Reference, Dictionary);
```

```
class THREAD : public Object
{
private:
int current_version; // most recent rev.
VOListReference the_list;

public:
```

```

THREAD(APL *theAPL);
THREAD(char *id);
virtual void Destroy(Boolean aborted=FALSE);
virtual Type *getDirectType();
int getCurrentVersionNum();
V_OBJECT *current();
V_OBJECT *version(int version_id);
void add_object(V_OBJECT *new_object);
void displayThreadVersions();
void displayThreadContents();
};

// Description -----
//
// Defines a THREAD class. The class COMPONENT is a derived class
// of Object (i.e. It is a persistent class). A thread may
// contain multiple versions of an COMPONENT, composite or
// configurations.
//
// Constructor
//
// Thread - APL
//
// ONTOS required constructor.
//
// Thread
//
// Constructs a thread with the given name.
//
// Public Members
//
// Destroy
//
// Used in lieu of a class destructor.
//
// getDirectType
//
// ONTOS required method to return the class type.
//
// getCurrentVersionNum
//
// Returns the version number of the vobject last add to the thread.
//
// current
//
// Returns a pointer to the current vobject in the thread.
//
// version
//
// Returns a pointer to a user designated version of a vobject.
//

```

```

// add_Object
//
// inserts a vobject into the thread.
//
// displayThreadVersions
//
// List the version numbers of vobjects contained in the thread.
//
// displayThreadContents
//
// Displays the version number and description of each vobject in the thread.
//
// End -----
#endif // _THREAD_H

```

```

// File Header -----
//.....:
//.Filename.....: thread.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in object file

static char thread.cxx.SccsId[] = "@(#)thread.cxx 1.3\t9/16/91";

// Contents -----
//
// THREAD::THREAD ONTOS constructor
// THREAD::THREAD new instance constructor
// THREAD::Destroy
// THREAD::getDirectType
// THREAD::getCurrentVersionNum
// THREAD::current
// THREAD::version
// THREAD::add_object
// THREAD::displayThreadVersions
// THREAD::displayThreadContents
//
// Description
//
// Implementation of class THREAD member functions.
//
// End -----

#include <GlobalEntities.h>
#include <stream.hxx>

#ifndef _THREAD_H
#include "thread.h"
#endif

#ifndef _VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

extern Type *THREAD_OType;
extern Type *V_OBJECT_OType;

THREAD::THREAD(APL *theAPL): (theAPL)

```

```
{
};
```

```
THREAD::THREAD(char *id): (id)
```

```

// Summary -----
//
// Constructs a persistent THREAD object. A thread contains
// a list of V_OBJECTS (objects which version), and maintains
// the most current version from that list of versioned
// objects.
//
// A thread is stored in the ONTOS database and is given
// visibility. Therefore, only one operator may generate any
// given thread.
//
// It is expected that Variations will inherit from threads
// with two distinctive bits of information:
//
// the thread from which it spawned -
// the version number from which it originated
//
// Parameter
//
// id
//
// passed to the ONTOS database and gives persistence and
// ONTOS visibility to that object
//
// Return Value
//
// a persistent THREAD in the ONTOS database
//
// End -----
```

```

{
  initDirectType(THREAD_OType);
  current_version=0;
  Dictionary *new_list=new Dictionary(OC_integer,
                                     V_OBJECT_OType,
                                     TRUE,FALSE);

  new_list →putObject();
  the_list.Reset(new_list, this);
  putObject();
};
```

```

void THREAD::Destroy(Boolean aborted)
{
  Destroy(aborted);
};
```

Type *THREAD::getDirectType()

```
// Summary -----  
//  
// returns the ONTOS Type for the prototype class.  
//  
// Return value  
//  
// A pointer to an ONTOS Type.  
//  
// End -----
```

```
{  
    return THREAD_OType;  
};
```

int THREAD::getCurrentVersionNum()

```
// Summary -----  
//  
// returns the current version number in the thread of  
// versioned objects  
//  
// Parameter  
//  
// N/A  
//  
// Return value  
//  
// An integer value representing the current version of the  
// operator/type (as defined by CAPS)  
//  
// End -----
```

```
{  
    return current_version;  
}
```

V_OBJECT *THREAD::current()

```
// Summary -----  
//  
// returns the current versioned object in the thread.  
//  
// Parameter  
//  
// N/A  
//  
// Return value  
//  
// A V_OBJECT pointer
```

```

//
// End -----
{
    Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

    V_OBJECT *mytempvo = (V_OBJECT*)(Entity*)(*temp_list)[current_version];

    return mytempvo;
};

```

V_OBJECT *THREAD::version(int the_version)

```

// Summary -----
//
// returns the desired version in the thread of versioned
// objects
//
// Parameter
//
// N/A
//
// Return value
//
// A V_OBJECT pointer
//
// End -----
{
    Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

    V_OBJECT *mytempvo = (V_OBJECT*)(Entity*)(*temp_list)[the_version];

    return mytempvo;
};

```

void THREAD::add_object(V_OBJECT *new_object)

```

// Summary -----
//
// adds a versioned_object to the thread, and updates the
// current_version attribute to reflect the newer version
//
// Parameter
//
// new_vobject
//
// V_OBJECT pointer

```



```

//
// Return Value
//
// N/A
//
// End -----

{
    if(!this)
    {
        cout << "<ERROR: cannot attach a v_object to a null THREAD>\n";
        return;
    }
    if(!new_object)
    {
        cout << "<ERROR: cannot insert a null v_object into a thread>\n";
        return;
    }
    else
    {
        current_version = current_version + 1;
        Dictionary *temp_list = (Dictionary*)the_list.Binding(this);
        temp_list → Insert(current_version,(Entity *)new_object);
        temp_list → putObject();
        putObject();
    }
};

```

```

void THREAD::displayThreadVersions()

```

```

// Summary -----
//
// Display the versions within a thread to stdout
//
// Parameter
//
// N/A
//
// Return Value
//
// N/A
//
// End -----

{
    Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

    DictionaryIterator next(temp_list);

    cerr << "THIS THREAD CONTAINS THE FOLLOWING VERSIONS:\n\n";

```

```

while(next.moreData())
{
    V_OBJECT *temp=(V_OBJECT *)(Entity *)next();
    temp->displayVersionNumber();
};

};

void THREAD::displayThreadContents()
{
    // Summary -----
    //
    // Displays the version and description of each versioned
    // object of a thread.
    //
    // NOT USED in current implementation of Design Database.
    //
    // Parameter
    //
    // N/A
    //
    // Return Value
    //
    // N/A
    //
    // End -----

    Dictionary *temp_list= (Dictionary*)the_list.Binding(this);

    DictionaryIterator next(temp_list);

    cerr << "THIS THREAD CONTAINS THE FOLLOWING VERSIONS:\n\n";
    while(next.moreData())
    {
        V_OBJECT *temp=(V_OBJECT *)(Entity *)next();
        temp->displayVersionNumber();
        cout << "\n";
        temp->getDescription();
        cout << "\n";
    };
};

```

```
// File Header -----
//.....:
//.Filename.....: tree.h
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----
```

```
#ifndef _TREE_H
#define _TREE_H
```

```
// SCCS ID follows: will compile to place date/time stamp in
// object file
```

```
static char tree_h_ScCsId[] = "@(#)tree.h 1.3\t9/16/91";
```

```
// Contents -----
//
// TREE HEADER
//
// Description
//
// Defines class TREE.
//
// End -----
```

```
// Interface Dependencies -----
```

```
#ifndef _TREENODE_H
#include "treenode.h"
#endif
```

```
// End Interface Dependencies -----
```

```
class TREE
{
private:
char *tree_name;
TREENODE * theTreeRootNode;

public:
TREE(TREENODE *,char *); // input list and resulting rootnode
void build_tree(TREENODE *,TREENODE_linkedlist);
TREENODE *find_treenode(TREENODE_linkedlist, char *);
};
```

```
// Description -----
```

```

//
// Defines the TREE class.
//
// Constructor
//
// Constructs a multiway tree from a linked list of nodes
// identified as operators from reading the subdirectory in
// TREENODE class. In this tree is one unique TREENODE -
// the "root". Once the root is identified, reference to
// the tree can be passed to other classes who can then deal
// individually with nodes in that tree through the TREENODE
// class.
//
// TREE
//
// constructs the tree given a TREENODE object and a character
// string pointer to the root operator.
//
// Public Members
//
// build_tree
//
// takes in the root TREENODE and a linked list of other
// generic operator nodes and builds the multiway tree using
// pointers and lists of children nodes.
//
// find_treenode
//
// given a character string of a TREENODE and a linked list of
// TREENODES, search the linked list and return a TREENODE
// pointer on a match. Return a NULL pointer if fails.
//
// End -----
#endif // _TREE.H

```

```
// File Header -----
//.....:
//.Filename.....: tree.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----
```

```
// SCCS ID follows: will compile to place date/time stamp in
// object file
```

```
static char tree_cxx_ScscId[] = "@(#)tree.cxx 1.3\t9/16/91";
```

```
// Contents -----
//
// TREE::TREE
// TREE::build_tree
// TREE::find_treenode
//
// Description
//
// IMPLEMENTS class TREE CONSTRUCTORS.
//
// End -----
```

```
// Interface Dependencies -----
```

```
#include <Database.h>
#include <stream.hxx>
```

```
extern "C--"
{
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <time.h>
}
```

```
#ifndef _TREE_H
#include "tree.h"
#endif
```

```
#ifndef _QUEUE_H
#include "queue.h"
```

```

#endif

#ifndef _NODESUPPORT_H
#include "nodesupport.h"
#endif

// ----- End Interface Dependencies -----

TREE::TREE(TREENODE *future_root,char *treename)

    // Summary -----
    //
    // Constructor
    //
    // Parameter
    //
    // future_root
    //
    // TREENODE pointer containing the future root of the multiway
    // tree
    //
    // treename
    //
    // character string - the same name as the root operator
    //
    // Return Value
    //
    // A constructed multiway tree reflecting the nodes which
    // exist in the subdirectory and which will be checked into
    // the design database
    //
    // End -----

{
    tree_name = new char [strlen(treename)+1];
    strcpy(tree_name,treename);
    theTreeRootNode = future_root;
}

TREENODE *TREE::find_treenode(TREENODE_linkedlist list_to_search,
                               char *node_name)

    // Summary -----
    //
    // find_treenode
    //
    // Parameter
    //
    // list_to_search
    //
    // a linked list of TREENODES

```

```

//
// node_name
//
// the operator/type to search for (i.e. - the name of the
// operators filename MINUS the .ps, .graph, .imp.psd!,
// .spec.psd!, .a extension
//
// Return Value
//
// TREENODE if found - NULL pointer if not found.
//
// End -----

{
    slist_iterator list_iterator(list_to_search);
    TREENODE *tnode;
    while (tnode=list_iterator())
        if (strcmp(tnode->getname(),node_name)==0)
            return tnode;
    return NULL;
}

void TREE::build_tree(TREENODE *root_node,TREENODE_linkedlist search_list)

// Summary -----
//
// Builds a multiway tree containing the nodes in the directory
// and information required to determine whether a new version
// of the node must be created in the ONTOS Design Database.
//
// Parameter
//
// root_node
//
// the unique TREENODE which is the root of this multiway tree
//
// search_list
//
// a list of operators in the subdirectory pointed to by the
// environment variable PROTOTYPE
//
// Return Value
//
// N/A
//
// End -----

{
    TREENODE *nodeptr;

```

```

TREENODE * temp_TREENODE_ptr;
TREENODE * new_TREENODE_ptr;
// create the queue inorder to construct the tree
TREENODE_queue tree_node_queue;
tree_node_queue.put(root_node);
// now the queue has the first TREENODE on it
while (!tree_node_queue.empty())
{
    temp_TREENODE_ptr = tree_node_queue.get();
    // now iterate through search_list , look for NODES whose associated
    // strings are "proper" superstrings of temp_TREENODE_ptr->operator_name
    // If they are create a TREENODE for these child nodes,
    // put the TREENODE in queue as well as in temp_TREENODE_ptr->children
    // list.
    slist_iterator OperatorPtr(search_list);
    while (nodeptr=OperatorPtr())
    {
        if (proper_super_NODE_check(nodeptr,temp_TREENODE_ptr->getname()))
            // create the new TREENODE
            {
                new_TREENODE_ptr = new TREENODE(nodeptr, temp_TREENODE_ptr);
                tree_node_queue.put(new_TREENODE_ptr);
                temp_TREENODE_ptr->insertChildNode(new_TREENODE_ptr);
            }
    }
}
}

```



```

// File Header -----
//.....:
//.Filename.....: treenode.h
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _TREENODE_H
#define _TREENODE_H

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char treenode_h_SccsId[] = "@(#)treenode.h 1.3\t9/16/91";

// Contents -----
//
// TREENODE HEADER
//
// Description
//
// Defines class TREENODE.
//
// End -----

// Interface Dependencies -----

#ifndef _QUEUE_H
#include "queue.h"
#endif

#ifndef _VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

// End Interface Dependencies -----
class TREENODE;

class TREENODE
{
private:
char *tree_node_name;
char *node_name;
long timestamp;
int level;
TREENODE_linkedlist ChildrenList;

```

```
TREENODE * ParentNode;
```

```
public:
```

```
TREENODE(char *,TREENODE *);
```

```
TREENODE(TREENODE *,TREENODE *);
```

```
void updatetimestamp(long time);
```

```
char *getname();
```

```
void insertChildNode(TREENODE *);
```

```
TREENODE_linkedlist getChildren();
```

```
TREENODE *getParentNode();
```

```
char *get_asc_time();
```

```
int getlevel();
```

```
long get_long_time();
```

```
void list_subtree();
```

```
void checkin_subtree(V_OBJECT *);
```

```
V_OBJECT *checkin_node(V_OBJECT *);
```

```
};
```

```
// Description -----  
//  
// TREENODE  
//  
// TREENODE  
//  
// Constructor - Builds a treenode to be a node plus pointer  
// information for building a multiway tree.  
//  
// updatetimestamp  
//  
// used to compare the most current filestamp of the group of  
// five files in a versioned object from the disk directory  
// pointed to by the CAPS environment variable PROTOTYPE to  
// the locktime of the matching versioned object stored  
// in the Design database.  
//  
// getname  
//  
// returns the character string name of the treenode.  
//  
// insertChildNode  
//  
// used to insert a node as a child of the current treenode  
//  
// getChildren  
//  
// returns the linked list of children of this node  
//  
// getParentNode  
//  
// returns the parent of this node  
//
```

```

// get_asc_time
//
// returns the ctime function for the treenode timestamp attribute
//
// get_level
//
// returns the integer level (0 = root, 1 is removed from root
// 1 level, etcetera
//
// get_long_time
//
// returns the timestamp from the treenode as a long that can
// be used in a comparison in the checkin_node function
//
// list_subtree
//
// used for debugging. Lists the multiway tree
//
// checkin_subtree
//
// after the multiway tree is built, this function launches the
// recursion which does the bulk of the work.
//
// checkin_node
//
// the function which compares the TREENODE (as read from
// disk) to threads in the database. If a match is found,
// locktimes are compared to timestamp and if timestamp is
// more recent, then a new versioned object is created for
// the database. All version links are set up in this
// function
//
// End Description -----

#endif // header file

```

```

// File Header -----
//.....:
//.Filename.....: treenode.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in
// object file

static char treenode_cxx_SccsId[] = "@(#)treenode.cxx 1.3\t9/16/91";

// Contents -----
//
// TREENODE::TREENODE
// TREENODE::TREENODE
// TREENODE::updatetimestamp
// TREENODE::getname
// TREENODE::insertChildNode
// TREENODE::getChildren
// TREENODE::getParentNode
// TREENODE::get_asc_time
// TREENODE::getlevel
// TREENODE::get_long_time
// TREENODE::list_subtree
// TREENODE::checkin_subtree
// TREENODE::checkin_node
//
// Description
//
// IMPLEMENTS class TREENODE CONSTRUCTORS and methods.
//
// End -----

// Interface Dependencies -----
#include <Directory.h>
#include <stream.hxx>

extern "C--"
{
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>

```

```

#include <time.h>
}
#ifndef _THREAD_H
#include "thread.h"
#endif

#ifndef _COMPONENT_H
#include "component.h"
#endif

#ifndef _TEXT_OBJECT_H
#include "text_object.h"
#endif

#ifndef _TREENODE_H
#include "treenode.h"
#endif

#ifndef _DDBDEFINES_H
#include "ddbdefines.h"
#endif

// ----- End Interface Dependencies -----

extern char *dirNamePtr;

TREENODE::TREENODE(char *name, TREENODE* future_parent)
{
    tree_node_name = new char[strlen(name)+1];
    strcpy(tree_node_name,name);
    level=0;
    timestamp = 0;
    ParentNode =future_parent;
}

TREENODE::TREENODE(TREENODE *inc_data, TREENODE* future_parent)
{
    tree_node_name = new char[strlen(inc_data->getname())+1];
    strcpy(tree_node_name,inc_data->getname());
    if ((future_parent)!=NULL)
        level=future_parent->getlevel()+1;
    else
        level=1;
    timestamp = inc_data->get_long_time();
    ParentNode = future_parent;
}

void TREENODE::updatetimestamp(long time)
{
    timestamp=time;
}

```

```

}

char *TREENODE::getname()
{
    return tree_node_name;
}

void TREENODE::insertChildNode(TREENODE *new_child)
{
    ChildrenList.insert(new_child);
}

TREENODE_Linkedlist TREENODE::getChildren()
{
    return ChildrenList;
}

TREENODE *TREENODE::getParentNode()
{
    return ParentNode;
}

char *TREENODE::get_asc_time()
{
    return ctime(&timestamp);
}

int TREENODE::getlevel()
{
    return level;
}

long TREENODE::get_long_time()
{
    return timestamp;
}

void TREENODE::list_subtree()
{
    slist_iterator ChildrenPtr(ChildrenList);
    TREENODE *tnode;
    for (;;) // recursive call inside infinite for loop
    {
        tnode = ChildrenPtr();
        if (tnode != NULL)
        {
            char *name = tnode->getname();
            int level = tnode->getlevel();
            char *asctime = tnode->get_asc_time();
            cout << level << "-->" << name << "time: " << asctime;
            tnode->list_subtree(); // preorder
        }
    }
}

```

```

    }
    else
        break; // breaks when end of list reached.
}
}

```

```

void TREENODE::checkin_subtree(V_OBJECT *new_parent)
{
    slist_iterator ChildrenPtr(ChildrenList);
    TREENODE *tnode;
    for (;;) // recursive call inside infinite for loop
    {
        tnode = ChildrenPtr();
        if (tnode != NULL)
        {
            V_OBJECT *parent;
            char *name = tnode->getname();
            cerr << "CHECKIN--> " << name << "\n";
            parent = tnode->checkin_node(new_parent);
            tnode->checkin_subtree(parent); //preorder
        }
        else
            break; // breaks when end of list reached.
    }
}

```

```

V_OBJECT *TREENODE::checkin_node(V_OBJECT *future_parent)
{
    ifstream psfile;
    ifstream graphfile;
    ifstream impfile;
    ifstream specfile;
    ifstream sourcefile;

    Boolean create_new_vobject = FALSE;
    char *psfilename;
    char *graphfilename;
    char *impfilename;
    char *specfilename;
    char *sourcefilename;

    char *COMPONENT_psfilename;
    char *COMPONENT_graphfilename;
    char *COMPONENT_impfilename;
    char *COMPONENT_specfilename;
    char *COMPONENT_sourcefilename;

    COMPONENT_psfilename = new char[strlen(tree_node_name)+LENGTH_PS_EXT+2];
    COMPONENT_graphfilename = new char[strlen(tree_node_name)+LENGTH_GRAPH_EXT+2];
    COMPONENT_impfilename = new char[strlen(tree_node_name)+LENGTH_IMP_EXT+2];
    COMPONENT_specfilename = new char[strlen(tree_node_name)+LENGTH_SPEC_EXT+2];

```

```

COMPONENT_sourcefilename = new char[strlen(tree_node_name)+LENGTH_SOURCE_EXT+2];

psfilename = new char[strlen(dirNamePtr) +strlen(tree_node_name)+LENGTH_PS_EXT+2];
graphfilename = new char[strlen(dirNamePtr)+strlen(tree_node_name)+LENGTH_GRAPH_EXT+2];
impfilename = new char[strlen(dirNamePtr)+strlen(tree_node_name)+LENGTH_IMP_EXT+2];
specfilename = new char[strlen(dirNamePtr)+strlen(tree_node_name)+LENGTH_SPEC_EXT+2];
sourcefilename = new char[strlen(dirNamePtr)+strlen(tree_node_name)+LENGTH_SOURCE_EXT+2];

strcpy(psfilename,dirNamePtr);
strcpy(graphfilename,dirNamePtr);
strcpy(specfilename,dirNamePtr);
strcpy(impfilename,dirNamePtr);
strcpy(sourcefilename,dirNamePtr);

strcat(psfilename,"/");
strcat(graphfilename,"/");
strcat(specfilename,"/");
strcat(impfilename,"/");
strcat(sourcefilename,"/");

strcat(psfilename,tree_node_name);
strcat(graphfilename,tree_node_name);
strcat(specfilename,tree_node_name);
strcat(impfilename,tree_node_name);
strcat(sourcefilename,tree_node_name);

strcat(psfilename,".ps");
strcat(graphfilename,".graph");
strcat(impfilename,".imp.psd1");
strcat(specfilename,".spec.psd1");
strcat(sourcefilename,".a");

strcpy(COMPONENT_psfilename,tree_node_name);
strcpy(COMPONENT_graphfilename,tree_node_name);
strcpy(COMPONENT_specfilename,tree_node_name);
strcpy(COMPONENT_impfilename,tree_node_name);
strcpy(COMPONENT_sourcefilename,tree_node_name);

strcat(COMPONENT_psfilename,".ps");
strcat(COMPONENT_graphfilename,".graph");
strcat(COMPONENT_impfilename,".imp.psd1");
strcat(COMPONENT_specfilename,".spec.psd1");
strcat(COMPONENT_sourcefilename,".a");

psfile.open(psfilename);
graphfile.open(graphfilename);
impfile.open(impfilename);
specfile.open(specfilename);
sourcefile.open(sourcefilename);

```



```

THREAD *threadPtr= (THREAD *)0;
V_OBJECT *vobjectPtr =(V_OBJECT *)0;
V_OBJECT *new_vobject = (V_OBJECT *)0;
long vobject_locktime = 0;
int versionNum = 1;

if (threadPtr ==((THREAD *)OC_lookup(tree_node_name)))
{
    vobjectPtr = threadPtr →current(); // return current vobject
    vobject_locktime = vobjectPtr →getLockTime(); // return locktime
    Boolean last_operation_was_checkin = vobjectPtr→get_last_operation();
    if (vobject_locktime < timestamp)
    {
        if (last_operation_was_checkin)
        {
            cout << "Last operation was VAA ... Preventing duplicates\n";
            return vobjectPtr;
        }
        else
        {
            versionNum = vobjectPtr → getVersionNumber() + 1;
            create_new_vobject = TRUE;
        }
    }
}
else
{
    threadPtr = new THREAD(tree_node_name);
    create_new_vobject = TRUE;
}

if (vobjectPtr)
{
    vobjectPtr→releaseLock();
    vobjectPtr→resetLastOpTrue();
    vobjectPtr→resetVisitedFlag(); // Just for good measure, before pass 2
    vobjectPtr→putObject();
}

if (create_new_vobject) // if compare says I need a new
{
    new_vobject= new V_OBJECT(versionNum);
    new_vobject→connect_vobject_to_thread(threadPtr);
    COMPONENT *new_COMPONENT=new COMPONENT();
    if (psfile)
    {
        TEXT_OBJECT *new_psfile_object= new TEXT_OBJECT();
        new_psfile_object→append(COMPONENT_psfilename,psfile);
        new_COMPONENT →addTextObject(new_psfile_object)
    }
}

```

```

if (graphfile)
{
    TEXT_OBJECT * new_graphfile_object=new TEXT_OBJECT();
    new_graphfile_object→append(COMPONENT_graphfilename,graphfile);
    new_COMPONENT →addTextObject(new_graphfile_object);
}
if (specfile)
{
    TEXT_OBJECT * new_specfile_object=new TEXT_OBJECT();
    new_specfile_object→append(COMPONENT_specfilename,specfile);
    new_COMPONENT →addTextObject(new_specfile_object);
}

if (impfile)
{
    TEXT_OBJECT * new_impfile_object=new TEXT_OBJECT();
    new_impfile_object→append(COMPONENT_impfilename,impfile);
    new_COMPONENT →addTextObject(new_impfile_object);
}

if (sourcefile)
{
    TEXT_OBJECT *new_sourcefile_object=new TEXT_OBJECT();
    new_sourcefile_object→append(COMPONENT_sourcefilename,sourcefile);
    new_COMPONENT →addTextObject(new_sourcefile_object);
}

new_vobject→addCOMPONENTNode(new_COMPONENT);
threadPtr→add_object(new_vobject);

if (future_parent)
{
    if (vobjectPtr)
        future_parent→deleteChildNode(vobjectPtr);
    future_parent→addChildNode(new_vobject);
    future_parent→putObject();
}
}
else
{
    if (future_parent)
    {
        future_parent→addChildNode(vobjectPtr);
        future_parent→putObject();
    }
}
psfile.close();
graphfile.close();
impfile.close();
specfile.close();

```

```

sourcefile.close();

if (create_new_vobject)
{
    new_vobject→setNodeName(getname());
    new_vobject→setParent(future_parent);
    return new_vobject; // return new version of vobject as parent
}
else
{
    vobjectPtr→setParent(future_parent);
    return vobjectPtr; // return old version of vobject as parent
}
}

```

```

// File Header -----
//.....:
//.Filename.....: versioned_object.h
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _VERSIONED_OBJECT_H
#define _VERSIONED_OBJECT_H

    // SCCS ID follows: will compile to place date/time stamp in object file

    static char versioned_object_h_SccsId[] = "@(#)versioned_object.h 1.3\t9/16/91";

#include <Object.h>
#include <List.h>
#include <Dictionary.h>
#include <Reference.h>
#include "ReferenceMacros.h"
#include <stream.hxx>

extern "C--"
{
#include <sys/time.h>
#include <sys/types.h>
#include <string.h>
}

#ifndef _THREAD_H
#include "thread.h"
#endif

#ifndef _COMPONENT_H
#include "component.h"
#endif

#define DEFAULT_VER 1

TypeCheckReference(DescReference, Reference, TEXT_OBJECT);
TypeCheckReference(COMPONENTObjReference, Reference, COMPONENT);
TypeCheckReference(ChildVObjReference, Reference, List);
TypeCheckReference(ThreadObjReference, Reference, THREAD);

class V_OBJECT : public Object
{
    TypeCheckReference(ParentObjReference, Reference, V_OBJECT);

```

private:

```
int theVersionNumber;
time_t creationDate;
time_t lockTime;
char *node_name;
char *creator;
char *worker;
Boolean visited; // for navigation through tree structure
Boolean last_op_checkin; // guards against double checkin
DescReference theDescriptionPtr;
ThreadObjReference theThreadPtr;
COMPONENTObjReference theCOMPONENTPtr;
ChildVObjReference theChildPtr;
ParentObjReference theParentPtr;
```

public:

```
V_OBJECT(APL *);
V_OBJECT(int= DEFAULT_VER);
void Destroy(Boolean aborted=FALSE);
Type *getDirectType();
void connect_vobject_to_thread(THREAD *);
void setParent(V_OBJECT *);
void setNodeName(char *);
char *getNodeName();
void getVObjName();
char *getName();
void resetVisitedFlag();
void setVisitedFlag();
Boolean getVisitedFlag();
void getVObjComponentsName();
void displayVersionNumber();
int getVersionNumber();
void dumpVObjSummary();
time_t setCreationDate();
time_t getCreationDate();
void setLock();
char *getWorker();
char *getCreator();
void setWorker();
void resetLastOpTrue();
void resetLastOpFalse();
Boolean get_last_operation(); // returns true if last op was checkin
int releaseLock();
time_t getLockTime();
void getDescription();
void listChildren();
void longlistOperatorNames();
void listOperatorNames('');
```

```

void updateDescription(char *, ifstream &);
void addCOMPONENTNode(COMPONENT *);
void deleteChildNode(V_OBJECT *);
void addChildNode(V_OBJECT *);
V_OBJECT *getParent();
COMPONENT *getCOMPONENT();
void dumpSubtree(char *);
void releaseLockSubtree();
List *getChildren();
Boolean getChildPtr();
Boolean checkoutCOMPONENTNode(char *);
~V_OBJECT() { Destroy(FALSE); };
};

// Description -----
// V_OBJECT
// V_OBJECT
//
// Constructors - builds a persistent object in the Ontos
// database.
//
// Destroy
//
// Required by Ontos. Every persistent object must have a
// destroy function.
//
// getDirectType
//
// Returns an Ontos Type
//
// connect_vobject_to_thread
//
// Connects a vobject to a thread bearing it's name.
//
// setParent
//
// Used to establish links (Transparent References as Ontos
// calls them) in the Design Database reflecting the
// decomposition of CAPS operators/types.
//
// setNodeName
//
// NodeName is maintained as a separate character string field.
//
// getNodeName
//
// get the shorter NodeName
//
// getVObjName
//
// Displays the versioned object's name to stdout

```

```

//
// getName
//
// returns the character string pointer of the Operator Name
//
// resetVisitedFlag
//
// resets visited to FALSE
//
// setVisitedFlag
//
// sets visited to TRUE
//
// getVisitedFlag
//
// returns the value of visited (Boolean)
//
// getVObjComponentsName
//
// display the different components in the Operator
//
// displayVersionNumber
//
// Display the version number of the V_OBJECT to stdout
//
// getVersionNumber
//
// return the int versionNumber
//
// dumpVObjSummary
//
// dump predetermined attribute values to the stdout/stderr
//
// setCreationDate
//
// gets the system time and stores it in CreationDate
//
// getCreationDate
//
// Displays the creation date as a 26 character ascii text
// string of the date and time
//
// setLock
//
// sets the lock to the system time
//
// getWorker
//
// returns the character string containing the workers name
//
// getCreator

```

```

//
// returns the character string containing the V_OBJECT
// creators name
//
// setWorker
//
// gets the UNIX UserPtr variable and stores the value of that
// variable into worker
//
// resetLastOpTrue
//
// Tells the system that the last operation on that V_OBJECT
// was a checkin. Prevents duplicate checkins from creating
// new versioned objects on each checkin
//
// resetLastOpFalse
//
// resets the last operation immediately following checkin
//
// get_last_operation
//
// returns a Boolean TRUE if last operation was an add
//
// releaseLock
//
// sets the lockTime back to 0 (epoch time) Sometime in 1969.
//
// getLockTime
//
// return the lockTime as a time_t (long) structure
//
// getDescription
//
// Display the V_OBJECT description (if one exists)
//
// listChildren
//
// list V_OBJECT's children
//
// longlistOperatorNames
//
// list V_OBJECT's children
//
// listOperatorNames
//
// gives the explicit name of the operator (to include
// path information from the root V_OBJECT
//
// updateDescription
//
// updates the versioned_objects description.

```



```

//
// addCOMPONENTNode
//
// adds an COMPONENT object to this V_OBJECT using the Ontos
// binding mechanism
//
// deleteChildNode
//
// removes an operator from the children list of the
// current V_OBJECT
//
// addChildNode
//
// adds an operator to the children list of the V_OBJECT
//
// getParent
//
// returns the parent V_OBJECT
//
// getCOMPONENT
//
// returns an COMPONENT pointer if one is contained in this
// V_OBJECT
//
// dumpSubtree
//
// attempts to rebuild files from versioned objects onto
// the UNIX subdirectory referenced by the UNIX variable
// PROTOTYPE
//
// releaseLockSubtree
//
// resets that node and every node under it to zero
// epoch time (sometime in 1969)
//
// getChildren
//
// returns a list of the children of the current V_OBJECT
//
// getChildPtr
//
// returns a Boolean TRUE if the Cardinality of the list
// referenced by the theChildPtr is > zero
//
// checkoutCOMPONENTNode
//
// attempts to rebuild the .ps, .graph, .imp.psdl, .spec.psdl
// and .a files of an operator/type stored in the Design
// Database
//
// End -----

```

#endif

```

// File Header -----
//.....:
//.Filename.....: versioned_object.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in object file

static char versioned_object_cxx_ScCsId[] = "0(#)versioned_object.cxx 1.3\t9/16/91";

// Contents -----
//
// V_OBJECT::V_OBJECT
// V_OBJECT::V_OBJECT
// V_OBJECT::Destroy
// V_OBJECT::getDirectType
// V_OBJECT::connect_vobject_to_thread
// V_OBJECT::setParent
// V_OBJECT::setNodeName
// V_OBJECT::getNodeName
// V_OBJECT::getVObjName
// V_OBJECT::getName
// V_OBJECT::resetVisitedFlag
// V_OBJECT::setVisitedFlag
// V_OBJECT::getVisitedFlag
// V_OBJECT::getVObjComponentsName
// V_OBJECT::displayVersionNumber
// V_OBJECT::getVersionNumber
// V_OBJECT::dumpVObjSummary
// V_OBJECT::setCreationDate
// V_OBJECT::getCreationDate
// V_OBJECT::setLock
// V_OBJECT::getWorker
// V_OBJECT::getCreator
// V_OBJECT::setWorker
// V_OBJECT::resetLastOpTrue
// V_OBJECT::resetLastOpFalse
// V_OBJECT::get_last_operation
// V_OBJECT::releaseLock
// V_OBJECT::getLockTime
// V_OBJECT::getDescription
// V_OBJECT::listChildren
// V_OBJECT::longlistOperatorNames
// V_OBJECT::listOperatorNames
// V_OBJECT::updateDescription

```

```

// V.OBJECT::addCOMPONENTNode
// V.OBJECT::deleteChildNode
// V.OBJECT::addChildNode
// V.OBJECT::getParent
// V.OBJECT::getCOMPONENT
// V.OBJECT::dumpSubtree
// V.OBJECT::releaseLockSubtree
// V.OBJECT::getChildren
// V.OBJECT::getChildPtr
// V.OBJECT::checkoutCOMPONENTNode
//
// Description
//
// methods for manipulating versioned objects
//
// End -----

// Interface Requirements -----

#include <GlobalEntities.h>
#include <Directory.h>

#ifndef _VERSIONED_OBJECT_H
#include "versioned_object.h"
#endif

#ifndef _TRACER_H
#include "tracer.h"
#endif

#ifndef __DDBDEFINES_H
#include "ddbdefines.h"
#endif

// Constructor //

// End Interface Requirements -----

extern Type *V.OBJECT_OType;
extern userPtr;

V.OBJECT::V.OBJECT(APL *theAPL) : Object(theAPL)

// Summary -----
//
// This is an activation constructor required by ONTOS.
// ONTOS calls the activation constructor anytime an object
// is brought into memory. Note the constructor passes
// theAPL to the base class APL constructor.

```

```

//
// Parameter
//
// theAPL
//
// A pointer to an APL (for Activation Parameter List) a
// structure.

{
    // empty by design
};

// End -----

// Constructor //

V_OBJECT::V_OBJECT(int versionNum)

    // Summary-----
    //
    // Parameter
    //
    // Functional Description
    //
{
    initDirectType(V_OBJECT_OType);
    theVersionNumber = versionNum;
    creationDate = setCreationDate();
    lockTime = 0;
    last_op_checkin = TRUE;
    visited = FALSE;
    creator = new char [strlen(userPtr)+1];
    strcpy(creator,userPtr);
    worker = (char *)0;
    node_name = (char *)0;
    theDescriptionPtr.initToNull();
    theCOMPONENTPtr.initToNull();
    theThreadPtr.initToNull();
    theParentPtr.initToNull();
    List *newChildList = new List(V_OBJECT_OType);
    newChildList → putObject();
    theChildPtr.Reset(newChildList, this);
    putObject();
};

// End -----

// Member Function (in lieu of destructor)//

void V_OBJECT::Destroy(Boolean aborted)

```

```

// Summary -----
//
// This one is semi tricky. Destroy redefined the Destroy
// function inherited from the class CleanupObj. Destroy()
// is used to delete CleanupObj objects and those of all its
// derived classes. In defining any class that is directly
// or indirectly derived from CleanupObj, provide the
// function Destroy(Boolean aborted) in place of a destructor
// if there is any special processing required when the
// object's memory is deallocated.
//
// Parameter
//
// aborted
//
// If Destroy() is called as a result of an abort, aborted
// is set to TRUE; if it is called to delete the object for
// other reasons, aborted is set to False.
//
// Functional Description
//
// CleanupObj in effect provides an audit trail of the
// creation of all stack-based instances of its derived
// classes, so that they can be cleanly deleted in the
// case of an abort during exception handling. Hence
// the Destroy function.
{
    Object::Destroy(aborted);
};

// End -----

// Member Function //

Type* V_OBJECT::getDirectType()
{
    return V_OBJECT.OType;
}

// Member Function //

void V_OBJECT::connect_vobject_to_thread(THREAD *threadPtr)
{
    theThreadPtr.Reset(threadPtr,this);
    putObject();
}

void V_OBJECT::setParent(V_OBJECT *parent)
{
    theParentPtr.Reset(parent,this);
}

```

```

    putObject();
}

void V_OBJECT::setNodeName(char *tree_node_name)
{
    char *token = (char *)0;
    token = strchr(tree_node_name, '.');
    if (token)
    {
        node_name = new char[strlen(token)+1];
        strcpy(node_name, token); // token in the subtree
        node_name++; // discard period (.)
    }
    else
    {
        node_name = new char [strlen(tree_node_name)+1];
        strcpy(node_name, tree_node_name); // must be root root.
    }
    putObject();
}

char *V_OBJECT::getNodeName()
{
    return node_name;
}

char * V_OBJECT::getName()
{
    char *name;
    Directory *directory;

    if(!this)
    {
        cerr << "<ERROR: cannot get the name of a null V_OBJECT>\n";
        return NULL;
    }
    if (!theThreadPtr)
    {
        cerr << "<ERROR: cannot display the name of a null thread>\n";
        return NULL;
    }
    else
    {
        THREAD *myThreadPtr = (THREAD*) theThreadPtr.Binding(this);
        name = myThreadPtr -> Name();
        OC_getNameComponents(name, &directory, &name);
        return name;
    }
}

```

// Member Function //

void V_OBJECT::getVObjName()

```
{
    char *name;
    Directory *directory;

    if(!this)
    {
        cerr << "<ERROR: cannot get the name of a null V_OBJECT>\n";
        return;
    }
    if(!theThreadPtr)
    {
        cerr << "<ERROR: cannot display the name of a null thread>\n";
        return;
    }
    else
    {
        THREAD *myThreadPtr = (THREAD*) theThreadPtr.Binding(this);
        name = myThreadPtr -> Name();
        OC_getNameComponents(name, &directory, &name);
        cout << name << "\n";
    }
}
```

// Member Function //

void V_OBJECT::getVObjComponentsName()

```
{ if(!this)
{
    cerr << "<ERROR: cannot get the names of a null V_OBJECT>\n";
    return;
}
if(!theCOMPONENTPtr)
{
    cerr << "<This v_object does not have an COMPONENT component>\n";
    return;
}
else
{
    COMPONENT *myCOMPONENTPtr = (COMPONENT*) theCOMPONENTPtr.Binding(this);
    myCOMPONENTPtr -> getComponentNames();
}
}
```

// Member Function //

void V_OBJECT::displayVersionNumber()

// Summary -----


```

        //
        // This function displays the version number of an object.

{
    cout << theVersionNumber << "\n";
};

int V_OBJECT::getVersionNumber()
{
    return theVersionNumber;
}

// End -----

// Member Function //

void V_OBJECT::dumpVObjSummary()
{
    // displayVersionNumber();
    cerr << "Date: ";
    cout << ctime(&creationDate);
    cerr << "Creator: ";
    cout << getCreator() << "\n";
    cerr << "Checked out out by ";
    if (worker)
    {
        cout << worker << "\n";
    }
    else
        cout << "NONE \n";
    cerr << "LockTime: ";
    if (!lockTime==0)
        cout << ctime(&lockTime);
    else
        cout << "NONE\n\n";
    cerr << "\n"
        << "Description\n"
        << "=====\n\n";
    getDescription();
}

// Member Function //

time_t V_OBJECT::setCreationDate()
{
    time_t mytloc=0;
    time_t theTime;
    return theTime = time(mytloc);
}

```

// Member Function //

```
void V_OBJECT::setLock()
{
    lockTime = setCreationDate();
}
```

// Member function //

```
char *V_OBJECT::getWorker()
{
    return worker;
}
```

// Member function //

```
char *V_OBJECT::getCreator()
{
    return creator;
}
```

// Member function //

```
void V_OBJECT::setWorker()
{
    char *temp_worker = new char [strlen(userPtr)+1];
    strcpy(temp_worker,userPtr);
    if (worker)
    {
        cerr << "resetworker --> from "<< getWorker() << " to "<< temp_worker << "\n";
        delete worker;
    }
    else
        cerr << "setworker --> to " << temp_worker << "\n";
    worker = new char[strlen(temp_worker)+1];
    strcpy(worker,temp_worker);
}
```

// Member Function //

```
void V_OBJECT::resetVisitedFlag()
{
    visited = FALSE;
}
```

```
void V_OBJECT::setVisitedFlag()
{
    visited = TRUE;
}
```

```
Boolean V_OBJECT::getVisitedFlag()
```

```

{
    return visited;
}

//Member Function //

void V_OBJECT::resetLastOpTrue()
{
    last_op_checkin = TRUE;
}

void V_OBJECT::resetLastOpFalse()
{
    last_op_checkin = FALSE;
}

//Member Function //

Boolean V_OBJECT::get_Last_operation()
{
    return last_op_checkin;
}

//Member Function //

int V_OBJECT::releaseLock()
{
    if (!worker)
    {
        last_op_checkin = TRUE;
        lockTime = 0;
        return SUCCESS;
    }

    if (strcmp(userPtr,getWorker())==0)
    {
        last_op_checkin = TRUE;
        lockTime = 0;
        delete worker;
        worker = (char *)0;
        return SUCCESS;
    }
    else
    {
        cerr << "<ERROR: Only " << getWorker() << " May unlock this object!...Aborting>\n";
        return FAILED;
    }
}

//Member Function //

```

```

time_t V_OBJECT::getCreationDate()
{
    return creationDate;
}

// Member Function //

time_t V_OBJECT::getLockTime()
{
    return lockTime;
}

// Member Function //

void V_OBJECT::getDescription()

    // Summary -----
    //
    // This function displays the description of an object
    //

{
    if(!this)
    {
        cout << "<ERROR: cannot get the description of a null V_OBJECT>\n";
        return;
    }
    if(!theDescriptionPtr)
    {
        cerr << "<This v_object does not have a description>\n";
        return;
    }
    else
    {
        TEXT_OBJECT *myTextObjectPtr =
            (TEXT_OBJECT*) theDescriptionPtr.Binding(this);
        myTextObjectPtr -> text(cout);
    }
}

// Member Function //

void V_OBJECT::updateDescription(char *fileName, ifstream& input_file_stream)
{
    if(!this)
    {
        cerr << "<ERROR: cannot update the description of a null V_OBJECT>\n";
        return;
    }
}

```

```

else
{
    if (strcmp(userPtr, getCreator()) == 0)
    {
        if (!theDescriptionPtr)
        {
            TEXT_OBJECT *textObjectPtr = new TEXT_OBJECT();
            textObjectPtr → append(fileName, input_file_stream);
            textObjectPtr → putObject();
            theDescriptionPtr.Reset(textObjectPtr, this);
        }
        else
        {
            TEXT_OBJECT *textObjectPtr =
                (TEXT_OBJECT*) theDescriptionPtr.Binding(this);
            textObjectPtr → resetTheText();
            textObjectPtr → append(fileName, input_file_stream);
        }
    }
    else
        cerr << "<ERROR: only " << getCreator() << " may update description. Aborting...>\n";
}
putObject();
}

```

// Member Function //

```

void V_OBJECT::addComponentNode(Component *myComponentPtr)
{
    if (!this)
    {
        cerr << "<ERROR: cannot set the COMPONENT node of a null V_OBJECT\n";
        return;
    }
    if (!myComponentPtr)
    {
        cerr << "<ERROR: cannot give to a V_OBJECT a null COMPONENT>\n";
        return;
    }
    theComponentPtr.Reset(myComponentPtr, this);
    putObject();
}

```

// Member Function //

```

void V_OBJECT::deleteChildNode(V_OBJECT *myV_ObjPtr)
{
    List *child_nodes = (List *)theChildPtr Binding(this);

    if (!this)
    {

```

```

        cerr << "<ERROR: cannot delete the child node of a null V_OBJECT\n";
        return;
    }
    if (!child_nodes)
    {
        cerr << "<ERROR: cannot remove a NULL child>\n";
        return;
    }
    long location = 0;
    if (child_nodes->isMember(myV_ObjPtr))
    {
        location = child_nodes->Index(myV_ObjPtr);
        child_nodes->Remove(location);
        child_nodes->putObject();
    }
}

// Member Function //

void V_OBJECT::addChildNode(V_OBJECT *myV_ObjPtr)
{
    List *child_nodes = (List *)theChildPtr.Binding(this);

    if (!this)
    {
        cerr << "<ERROR: cannot set the child node of a null V_OBJECT\n";
        return;
    }
    if (!child_nodes)
    {
        cerr << "<ERROR: cannot give to a V_OBJECT a null child>\n";
        return;
    }
    child_nodes->Insert(myV_ObjPtr);
    child_nodes->putObject();
}

// Member Function //
V_OBJECT *V_OBJECT::getParent()
{
    return (V_OBJECT *) (Entity *) theParentPtr.Binding(this);
}

COMPONENT *V_OBJECT::getCOMPONENT()
{
    return (COMPONENT *) (Entity *) theCOMPONENTPtr.Binding(this);
}

void V_OBJECT::listChildren()
{

```

```

if(!this)
{
    cerr << "<ERROR: can not dump children of a null V_OBJECT!>\n";
    return;
}

if(!theChildPtr)
{
    cerr << "<ERROR: can not print children of a null childPtr!>\n";
    return;
}
else
{
    List *child_nodes = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_nodes);
    V_OBJECT *cnode;

    while(ChildrenPtr.moreData())
    {
        cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
        cerr << "Operator: ";
        cout << cnode->getNodeName();
        //*****
        // following for loop provides spacing...
        // *****
        int i=0;
        for (i=0;i<(PRINT_VERSION_LOCATION-strlen(cnode->getNodeName()));i++)
            cout << " ";
        cerr << "Version: ";
        cout << cnode->getVersionNumber();
        cout << "\n";
        time_t locktime = cnode->getLockTime();
        cerr << "Locktime is: " << ctime(&locktime) << "\n";
    }
    return;
}
}

void V_OBJECT::longlistOperatorNames()
{
    if(!this)
    {
        cerr << "<ERROR: can not dump operators of a null V_OBJECT!>\n";
        return;
    }

    if(!theChildPtr)
    {
        cerr << "<ERROR: can not print list of a null childPtr!>\n";
    }
}

```

```

        return;
    }
else
{
    List *child_nodes = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_nodes);
    V_OBJECT *cnode;
    while(ChildrenPtr.moreData())
    {
        cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
        // if (cnode->getChildPtr())
        // {
        //     cnode-> longlistOperatorNames();
        // }
        cerr << "Operator: ";
        cout << cnode->getNodeName();
        //*****
        // following for loop provides spacing...
        // *****
        int i=0;
        for (i=0;i<(PRINT_VERSION_LOCATION-strlen(cnode->getNodeName()));i++)
            cout << " ";
        cerr << "\nVersion: ";
        cout << cnode->getVersionNumber();
        cout << "\n";
        time_t locktime = cnode->getLockTime();
        cerr << "Locktime is: " << ctime(&locktime) << "\n";
    }
    return;
}
}

void V_OBJECT::listOperatorNames()
{
    Boolean node_was_visited = FALSE;
    if (!this)
    {
        cerr << "<ERROR: can not dump children of a null V_OBJECT!>\n";
        return;
    }
    if (!theChildPtr)
    {
        cerr << "<ERROR: can not print children of a null childPtr!>\n";
        return;
    }
else
{
    List *child_nodes = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_nodes);
    V_OBJECT *cnode;

```



```

int index = 1;
while(ChildrenPtr.moreData())
{
    cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
    node_was_visited = cnode->getVisitedFlag();
    if (cnode->getChildPtr())
    {
        cnode->listOperatorNames();
    }
    if (node_was_visited && index > 1)
    {
        cnode->resetVisitedFlag();
        cnode->putObject();
        break;
    }
    else
    {
        cnode->resetVisitedFlag();
        cnode->putObject();
    }
    if (index == 1)
        cnode->setVisitedFlag();
    index++;
    cerr << "Operator: ";
    cout << cnode->getName();
    //*****
    // following for loop provides spacing...
    // *****
    // int i=0;
    // for (i=0;i<(PRINT_VERSION_LOCATION-strlen(cnode->getName()));i++)
    // cout << " ";
    cerr << "\nVersion: ";
    cout << cnode->getVersionNumber();
    cout << "\n";
    time_t locktime = cnode->getLockTime();
    cerr << "Locktime is: " << ctime(&locktime) << "\n";
}
resetVisitedFlag();
putObject();
return;
}

void V_OBJECT::releaseLockSubtree()
{
    List *child_nodes = (List *) theChildPtr.Binding(this);
    ListIterator ChildrenPtr(child_nodes);
    V_OBJECT *cnode;

    while(ChildrenPtr.moreData())
    {

```

```

cnode = (V_OBJECT *) (Entity *) ChildrenPtr();
if (cnode→releaseLock())
    cnode→putObject();
else
{
    cerr << "<Error while unlocking " << cnode→getName() << " Aborting...>\n";
    break; // should try to unlock other siblings and their children.
}
if (cnode→getChildPtr())
{
    cnode→ releaseLockSubtree();
}
}
return;
}

```

```

void V_OBJECT::dumpSubtree(char *file_write_option)
{
    Boolean node_was_visited = FALSE;
    Boolean file_operation_successful = FALSE;
    Boolean write_operation = FALSE;
    if (strcmp(file_write_option,"w")==0 || strcmp(file_write_option,"W")==0)
        write_operation = TRUE;
    if (!this)
    {
        cerr << "<ERROR: can not dump children of a null V_OBJECT!>\n";
        return;
    }
    if (!theChildPtr)
    {
        cerr << "<ERROR: can not print children of a null childPtr!>\n";
        return;
    }
    else
    {
        List *child_nodes = (List *) theChildPtr.Binding(this);
        ListIterator ChildrenPtr(child_nodes);
        V_OBJECT *cnode;
        int index = 1;
        while (cnode = (V_OBJECT *) (Entity *)ChildrenPtr())
        {
            node_was_visited = cnode→getVisitedFlag();
            if (cnode→getChildPtr())
            {
                cnode→ dumpSubtree(file_write_option);
            }
            if (node_was_visited && index > 1)
            {
                cnode→resetVisitedFlag();
                cnode→putObject();
                break;
            }
        }
    }
}

```

```

    }
    else
    {
        cnode→resetVisitedFlag();
        cnode→putObject();
    }
    if (index == 1)
        cnode→setVisitedFlag();
    else
    {
        cnode→resetVisitedFlag();
        cnode→putObject();
    }
    index++;
    long cobject_locktime = 0;
    cobject_locktime = cnode→getLockTime();
    if (cobject_locktime > 0) // prevent checkout
    {
        if (strcmp(file_write_option, "w") == 0) // change "w" to "r"
        {
            cerr << "<ERROR: Module " << cnode→getNodeName()
                << " locked by : " << cnode→getWorker()
                << " Resetting write option to read-only>\n";
            strcpy(file_write_option, "r");
        }
        cerr << "<Caution: " << cnode→getNodeName()
            << " is locked.> \n" << "Date Locked: "
            << ctime(&cobject_locktime)
            << "Node checked out in read-only mode\n";
    }
    cerr << "NODENAME ---> " << cnode→getNodeName() << "\n";
    cerr << "Version: " << cnode→getVersionNumber() << "\n\n";
    file_operation_successful = cnode→checkoutCOMPONENTNode(file_write_option);
    if ((file_operation_successful) &&
        ((strcmp(file_write_option, "w") == 0) ||
         (strcmp(file_write_option, "w") == 0)))
    {
        cnode→setLock();
        cnode→setWorker();
        cnode→resetLastOpFalse();
        cnode→putObject();
    }
    if (write_operation)
        strcpy(file_write_option, "w");
}
resetVisitedFlag();
putObject();
return;
}
}

```

```

List * V_OBJECT::getChildren()
{
    List *temp_list = (List *)theChildPtr.Binding(this);
    if (temp_list→Cardinality() > 0)
        return temp_list;
    else
        return NULL;
}

Boolean V_OBJECT::getChildPtr()
{
    List *temp_list = (List *)theChildPtr.Binding(this);
    if (temp_list→Cardinality() > 0)
        return TRUE;
    else
        return FALSE;
}

// Member Function //

Boolean V_OBJECT::checkoutCOMPONENTNode(char *file_write_option)
{
    if (!this)
    {
        cerr << "<ERROR: cannot checkout COMPONENT nodes of a null V_OBJECT\n";
        return FAILED;
    }
    if (!theCOMPONENTPtr)
    {
        cerr << "<ERROR: cannot get the source code from a null COMPONENT>\n";
        return FAILED;
    }
    else
    {
        COMPONENT *myCOMPONENTPtr = (COMPONENT*) theCOMPONENTPtr.Binding(this);
        return (myCOMPONENTPtr → getComponentSource(file_write_option));
    }
}

```

```

// File Header -----
//.....:
//.Filename.....: vobjectfunc.h
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

#ifndef _VOBJECTFUNC_H
#define _VOBJECTFUNC_H

    // SCCS ID follows: will compile to place date/time stamp in object file

    static char vobjectfunc_h_SccsId[] = "@(#)vobjectfunc.h 1.3\t9/16/91";

// Contents -----
//
// VOBJECTFUNC
//
// Description
//
// Defines functions manipulating Versioned Objects
// (Operators) as called by main()
//
// End -----

// Interface Dependencies -----
//
// NONE
//
// End Interface Dependencies -----

void list_operators_func(int, char *, char *, char *);
void update_vobject_desc_func(int, char *, char *, char *, char *);
void get_vobject_desc_func(int, char *, char *, char *);
void get_vobject_date_func(int, char *, char *, char *);
void get_vobject_versions_func(int, char *, char *);
void get_vobject_lock_func(int, char *, char *, char *);
void get_vobject_version_func();
void dump_vobject_summary_func(int, char *, char *, char *);
void get_vobject_psf_file_func(int, char *, char *, char *, char *);
void get_vobject_graphfile_func(int, char *, char *, char *, char *);
void get_vobject_impfile_func(int, char *, char *, char *, char *);
void get_vobject_specfile_func(int, char *, char *, char *, char *);
void get_vobject_sourcefile_func(int, char *, char *, char *, char *);
void dump_vobject_files_func(int, char *, char *, char *, char *);
void dump_vobject_tree_func(int, char *, char *, char *, char *);

```

```

void long_list_children_func(int,char *, char *, char *);
void long_list_parents_func(int,char *, char *, char *);
void long_list_operators_func(int, char *, char *, char *);
void release_operator_lock_func(int, char *, char *, char *);
void release_subtree_lock_func(int, char *, char *, char *);
void add_vobject_and_subtree_func(int , char *, char *);

```

```

// Description -----
//
// list_operators_func
//
// Provides a list of operators associated with a
// subtree of a designated versioned_object.
//
// update_vobject_desc_func
//
// Updates the description text of a designated versioned
// object.
//
// get_vobject_desc_func
//
// Provides a description of the designated versioned object.
//
// get_vobject_date_func
//
// Displays the date that the versioned object was created.
//
// get_vobject_versions_func
//
// Checks the thread and displays the different versions.
//
// get_vobject_lock_func
//
// Displays a 26 character text entry reflecting the time
// and date that the versioned object was locked.
//
// get_vobject_version_func
//
// returns the version of the versioned object instance.
//
// dump_vobject_summary_func
//
// displays predetermined attribute fields to stdout/stderr.
//
// get_vobject_psfile_func
//
// rebuilds the CAPS postscript file to the PROTOTYPE
// directory.
//
// get_vobject_graphfile_func
//

```

```

// rebuilds the CAPS graph file to the PROTOTYPE directory.
//
// get_vobject_impfile_func
//
// rebuilds the CAPS implementation file to the PROTOTYPE directory.
//
// get_vobject_specfile_func
//
// rebuilds the CAPS specification file to the PROTOTYPE directory.
//
// get_vobject_sourcefile_func
//
// rebuilds the CAPS source file to the PROTOTYPE directory.
//
// dump_vobject_files_func
//
// rebuilds all of the above files (of one operator) to the
// PROTOTYPE directory.
//
// dump_vobject_tree_func
//
// rebuilds all existing TEXT_OBJECT attributes to the PROTOTYPE
// directory for the entire decomposition of the root operator
// down to the component operators. May be dumped in "read only"
// or "read/write". When dumped, all versioned objects are
// locked for modification by other users.
//
// long_list_children_func
//
// lists the node name and version number of children.
//
// long_list_parents_func
//
// lists the most current parent (and that parents siblings).
//
// long_list_operators_func
//
// lists the node name and version number of a node's children.
//
// release_operator_lock_func
//
// reset the locktime of a versioned operator.
//
// release_subtree_lock_func
//
// reset the locktime of a versioned operator and all children
// of that versioned operator.
//
// add_vobject_and_subtree_func
//
// the reverse of a dumping vobjects. this checks versioned

```

```
// objects back into the database, versioning them if necessary.  
//  
// End Description -----  
  
*endif // end vobjectfunc header function
```



```

// File Header -----
//.....:
//.Filename.....: vobjectfunc.cxx
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Author.....: Garry Lewis
//.....: Drew Dwyer
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

// SCCS ID follows: will compile to place date/time stamp in object file

static char vobjectfunc_cxx_SccsId[] = "@(#)vobjectfunc.cxx 1.3\t9/16/91";

// Contents -----
//
// list_operators_func
// update_vobject_desc_func
// get_vobject_desc_func
// get_vobject_date_func
// get_vobject_versions_func
// get_vobject_lock_func
// get_vobject_version_func
// dump_vobject_summary_func
// get_vobject_psfile_func
// get_vobject_graphfile_func
// get_vobject_impfile_func
// get_vobject_specfile_func
// get_vobject_sourcefile_func
// dump_vobject_files_func
// dump_vobject_tree_func
// long_list_children_func
// long_list_parents_func
// long_list_operators_func
// release_operator_lock_func
// release_subtree_lock_func
// add_vobject_and_subtree_func
//
// Description
//
// this contains the functions for manipulating versioned objects
//
// End -----

// Interface Dependencies -----
//

#include <stream.hxx>
#include <List.h>

```

```

#include <Directory.h>

extern "C--"{
#include <stdlib.h>
}

#ifndef _DIRECTORY_H
#include "directory.h"
#endif

#ifndef _TREE_H
#include "tree.h"
#endif

#ifndef _TREENODE_H
#include "treenode.h"
#endif

#ifndef _PROTOTYPE_H
#include "prototype.h"
#endif

#ifndef _COMPONENT_H
#include "component.h"
#endif

#ifndef _VOBJECT_H
#include "versioned_object.h"
#endif

#ifndef _THREAD_H
#include "thread.h"
#endif

#ifndef _VOBJECTFUNC_H
#include "vobjectfunc.h"
#endif

#ifndef _DDBDEFINES_H
#include "ddbdefines.h"
#endif

//
// End Interface Dependencies -----

extern Type *THREAD_OType;
extern Directory *prototype_dir;
PROTOTYPE *prototypeptr;
THREAD *threadPtr;
COMPONENT *COMPONENTPtr;

```

```
long vobject_locktime = 0;
```

```
void list_operators_func(int number_arguments, char *proto_name,
                        char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);
    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr;
                    vobjectPtr = threadPtr->current();
                    cerr << "Operator:  ";
                    cout << vobjectPtr->getName();
                    cerr << "\nVersion:  ";
                    cout << vobjectPtr->getVersionNumber();
                    cout << "\n";
                    time_t locktime = vobjectPtr->getLockTime();
                    cerr << "Locktime is:  " << ctime(&locktime) << "\n";

                    vobjectPtr->listOperatorNames();
                }
            }
            else
            {
                cout << "<Error getting thread in LIST OPERATORS:>\n";
            }
        }
        else
        {
            cout << "<Error getting Prototype in LIST OPERATORS:>\n";
        }
    }
    break;
    case 3:
        prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
        if (prototypeptr)
        {
            threadPtr = (THREAD *)OC_lookup(operator_name);
            if (threadPtr)
            {
                V_OBJECT *vobjectPtr;
                vobjectPtr = threadPtr->version(atoi(versionstr));
                cerr << "Operator:  ";
                cout << vobjectPtr->getName();
            }
        }
    }
}
```

```

        cerr << "\nVersion: ";
        cout << vobjectPtr->getVersionNumber();
        cout << "\n";
        time_t locktime = vobjectPtr->getLockTime();
        cerr << "Locktime is: " << ctime(&locktime) << "\n";
        vobjectPtr->listOperatorNames();
    }
    else
    {
        cout << "<Error getting thread in GET_VOBJECT_DESC:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_DESC:>\n";
}
break;
default:
    cout << "<ERROR: invalid number args for get vobject description>\n";
}
}

```

```

void update_vobject_desc_func(int number_arguments, char *proto_name,
                              char *operator_name, char *filename,
                              char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    ifstream description_file;
    description_file.open(filename);
    if (!description_file)
        cout << "<Description File not Found! >\n"
              << "<Abnormal terminate from update_vobject_desc_func>\n";
    else
    {
        switch (number_arguments)
        {
            case 3:
                prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
                if (prototypeptr)
                {
                    threadPtr = (THREAD *)OC_lookup(operator_name);
                    if (threadPtr)
                    {
                        V_OBJECT *vobjectPtr;
                        vobjectPtr = threadPtr->current();
                        vobjectPtr->updateDescription(filename, description_file);
                    }
                }
            else

```

```

        {
            cout << "<Error getting thread in UPDATE_VOBJECT_DESC:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in UPDATE_VOBJECT_DESC:>\n";
    }
    break;
case 4:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr->version(atoi(versionstr));
            vobjectPtr->updateDescription(filename,description_file);
        }
        else
        {
            cout << "<Error getting thread in UPDATE_VOBJECT_DESC:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in UPDATE_VOBJECT_DESC:>\n";
    }
    break;
default:
    cout << "<ERROR: invalid number args for update vobject description>\n";
}
}
}

```

```

void get_vobject_desc_func(int number_arguments,char *proto_name,
                           char *operator_name,char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)

```

```

        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr→current();
            vobjectPtr →getDescription();
        }
        else
        {
            cout << "<Error getting thread in GET_VOBJECT_DESC:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET_VOBJECT_DESC:>\n";
    }
    break;
case 3:
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypePtr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr→version(atoi(versionstr));
            vobjectPtr →getDescription();
        }
        else
        {
            cout << "<Error getting thread in GET_VOBJECT_DESC:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET_VOBJECT_DESC:>\n";
    }
    break;
default:
    cout << "<ERROR: invalid number args for get vobject description>\n";
}
}

```

```

void release_subtree_lock_func(int number_arguments, char *proto_name,
                               char *operator_name, char *versionstr)

```

```

{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:

```

```

prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
if (prototypeptr)
{
    threadPtr = (THREAD *)OC_lookup(operator_name);
    if (threadPtr)
    {
        V_OBJECT *vobjectPtr;
        vobjectPtr = threadPtr→current();

        if (vobjectPtr→releaseLock())
        {
            vobjectPtr→putObject();
            vobjectPtr→releaseLockSubtree();
        }
        else
            cerr << "<Error:  Couldn't unlock " << vobjectPtr→getName()
                << " Aborting releaseLock for rest of subtree>\n";
    }
    else
    {
        cout << "<Error getting thread in SUBTREE_RELEASE_LOCK:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in SUBTREE_RELEASE_LOCK:>\n";
}
break;
case 3:
prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
if (prototypeptr)
{
    threadPtr = (THREAD *)OC_lookup(operator_name);
    if (threadPtr)
    {
        V_OBJECT *vobjectPtr;
        vobjectPtr = threadPtr→version(atoi(versionstr));
        if (vobjectPtr→releaseLock())
        {
            vobjectPtr→putObject();
            vobjectPtr→releaseLockSubtree();
        }
        else
            cerr << "<Error:  Couldn't unlock " << vobjectPtr→getName()
                << " Aborting releaseLock for rest of subtree>\n";
    }
    else
    {
        cout << "<Error getting thread in SUBTREE_RELEASE_LOCK:>\n";
    }
}
}

```

```

        else
        {
            cout << "<Error getting Prototype in SUBTREE_RELEASE_LOCK:>\n";
        }
        break;
    default:
        cout << "<ERROR: invalid number args for subtree release lock>\n";
    }
}

void release_operator_lock_func(int number_arguments, char *proto_name,
                                char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr;
                    vobjectPtr = threadPtr->current();
                    if (vobjectPtr->releaseLock())
                        vobjectPtr->putObject();
                    else
                    {
                        cerr << "<Error: Couldn't unlock "<<vobjectPtr->getName()
                            << " Aborting release lock>\n";
                    }
                }
                else
                {
                    cout << "<Error getting thread in RELEASELOCK:>\n";
                }
            }
            else
            {
                cout << "<Error getting Prototype in RELEASELOCK:>\n";
            }
            break;
        case 3:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {

```



```

        V_OBJECT *vobjectPtr;
        vobjectPtr = threadPtr->version(atoi(versionstr));
        if (vobjectPtr->releaseLock())
            vobjectPtr->putObject();
        else
            cerr << "<Error:  Couldn't unlock "<<vobjectPtr->getName()
                << " Aborting release lock>\n";
    }
    else
    {
        cout << "<Error getting thread in RELEASELOCK:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in RELEASELOCK:>\n";
}
break;
default:
    cout << "<ERROR: invalid number args for release lock>\n";
}
}

```

```

void get_vobject_date_func(int number_arguments, char *proto_name,
                           char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);
    time_t creation_date;

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr;
                    vobjectPtr = threadPtr->current();
                    creation_date = vobjectPtr->getCreationDate();
                    cout << ctime(&creation_date) << "\n";
                }
            }
            else
            {
                cout << "<Error getting thread in GET_VOBJECT_DATE:>\n";
            }
        }
    }
    else

```

```

        {
            cout << "<Error getting Prototype in GET_VOBJECT_DATE:>\n";
        }
        break;
    case 3:
        prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
        if (prototypeptr)
        {
            threadPtr = (THREAD *)OC_lookup(operator_name);
            if (threadPtr)
            {
                V_OBJECT *vobjectPtr;
                vobjectPtr = threadPtr->version(atoi(versionstr));
                creation_date = vobjectPtr->getCreationDate();
                cout << ctime(&creation_date) << "\n";
            }
            else
            {
                cout << "<Error getting thread in GET_VOBJECT_DATE:>\n";
            }
        }
        else
        {
            cout << "<Error getting Prototype in GET_VOBJECT_DATE:>\n";
        }
        break;
    default:
        cout << "<ERROR: invalid number args for get vobject date>\n";
    }
}

```

```

void get_vobject_versions_func(int number_arguments, char *proto_name,
                               char *operator_name)

```

```

{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    threadPtr->displayThreadVersions();
                }
            }
            else
            {

```

```

        cout << "<Error getting thread in GET_VOBJECT_VERSIONS:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_VERSIONS:>\n";
}
break;
default:
    cout << "<ERROR: invalid number args for get vobject VERSIONS>\n";
}
}

```

```

void get_vobject_lock_func(int number_arguments, char *proto_name,
                           char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);
    time_t lock_time;

    switch (number_arguments)
    {
    case 2:
        prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
        if (prototypeptr)
        {
            threadPtr = (THREAD *)OC_lookup(operator_name);
            if (threadPtr)
            {
                V_OBJECT *vobjectPtr;
                vobjectPtr = threadPtr->current();
                lock_time = vobjectPtr->getLockTime();
                cerr << "Locktime: ";
                cout << ctime(&lock_time) << "\n";
            }
        }
        else
        {
            cout << "<Error getting thread in GET_VOBJECT_LOCK:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET_VOBJECT_LOCK:>\n";
    }
    break;
    case 3:
        prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
        if (prototypeptr)
        {
            threadPtr = (THREAD *)OC_lookup(operator_name);

```

```

        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr->version(atoi(versionstr));
            lock_time = vobjectPtr->getLockTime();
            cerr << "Locktime: ";
            cout << ctime(&lock_time) << "\n";
        }
        else
        {
            cout << "<Error getting thread in GET_VOBJECT_LOCK:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET_VOBJECT_LOCK:>\n";
    }
    break;
default:
    cout << "<ERROR: invalid number args for get vobject lock>\n";
}

}

void get_vobject_version_func()
{
    cout << "Not implemented. Unclear specs for get version of vobject\n";
}

void dump_vobject_summary_func(int number_arguments, char *proto_name,
                              char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr;
                    vobjectPtr = threadPtr->current();
                    vobjectPtr->dumpVObjSummary();
                }
            }
            else
            {

```

```

        cout << "<Error getting thread in GET_VOBJECT_SUMMARY:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_SUMMARY:>\n";
}
break;
case 3:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr->version(atoi(versionstr));
            vobjectPtr->dumpVObjSummary();
        }
        else
        {
            cout << "<Error getting thread in GET_VOBJECT_SUMMARY:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET_VOBJECT_SUMMARY:>\n";
    }
    break;
default:
    cout << "<ERROR: invalid number args for get vobject summary>\n";
}
}

```

```

void get_vobject_pfile_func(int number_arguments, char *proto_name,
                           char *operator_name, char *file_write_option, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr;

```

```

vobjectPtr = threadPtr→current();
vobject_locktime = vobjectPtr→getLockTime();
if (vobject_locktime>0) // prevent checkout
{
    if (strcmp(file_write_option,"w")==0) // change "w" to "r"
    {
        cerr << "<ERROR: Module " << vobjectPtr→getNodeName()
            << " locked by : " << vobjectPtr→getWorker()
            << " Resetting write option to read-only>\n";
        strcpy(file_write_option,"r");
    }
    cerr << "<Caution: " << vobjectPtr→getNodeName()
        << " is locked.> \n" << "Date Locked: "
        << ctime(&vobject_locktime)
        << "operator files checked out in read-only mode\n";
}
else
    cerr << "NODENAME ---> " << vobjectPtr→getNodeName() << "\n";
COMPONENTPtr = vobjectPtr→getCOMPONENT();
if ((COMPONENTPtr→getPSfile(file_write_option)) &&
    ((strcmp(file_write_option,"w")==0) ||
     (strcmp(file_write_option,"r")==0)))
{
    vobjectPtr → setLock(); // set root lock
    vobjectPtr → setWorker();
    vobjectPtr → resetLastOpFalse();
    vobjectPtr→putObject();
}
}
else
{
    cout << "<Error getting thread in GET_VOBJECT_PS:>\n";
}
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_PS:>\n";
}
}
break;
case 3:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr→version(atoi(versionstr));
            vobject_locktime = vobjectPtr→getLockTime();
            if (vobject_locktime>0) // prevent checkout
            {

```

```

        if (strcmp(file_write_option,"w")==0) // change "w" to "r"
        {
            cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                << " locked by : " << vobjectPtr->getWorker()
                << " Resetting write option to read-only>\n";
            strcpy(file_write_option,"r");
        }
        cerr << "<Caution:  " << vobjectPtr->getNodeName()
            << " is locked.> \n"
            << "Date Locked:  " << ctime(&vobject.locktime)
            << "operator files checked out in read-only mode\n";
    }
    else
        cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
    if ((strcmp(file_write_option,"w")==0) ||
        (strcmp(file_write_option,"w")==0))
    {
        vobjectPtr->setLock(); // set root lock
        vobjectPtr->setWorker();
        vobjectPtr->resetLastOpFalse();
        vobjectPtr->putObject();
    }
    COMPONENTPtr = vobjectPtr->getCOMPONENT();
    COMPONENTPtr->getPSfile(file_write_option);
}
else
{
    cout << "<Error getting thread in GET.VOBJECT.PS:>\n";
}
}
else
{
    cout << "<Error getting Prototype in GET.VOBJECT.PS>";
}
break;
default:
    cout << "<ERROR: invalid number args for get vobject PS>\n";
}
}

void get_vobject_graphfile_func(int number_arguments, char *proto_name,
                                char *operator_name, char *file_write_option, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {

```

```

case 2:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr->current();
            vobject_locktime = vobjectPtr->getLockTime();
            if (vobject_locktime>0) // prevent checkout
            {
                if (strcmp(file_write_option,"w")==0) // change "w" to "r"
                {
                    cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                        << " locked by : " << vobjectPtr->getWorker()
                        << " Resetting write option to read-only>\n";
                    strcpy(file_write_option,"r");
                }
                cerr << "<Caution: " << vobjectPtr->getNodeName()
                    << " is locked.> \n" << "Date Locked: "
                    << ctime(&vobject_locktime)
                    << "operator files checked out in read-only mode\n";
            }
            else
                cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
            COMPONENTPtr = vobjectPtr->getCOMPONENT();
            if ((COMPONENTPtr->getGRAPHfile(file_write_option)) &&
                ((strcmp(file_write_option,"w")==0) ||
                 (strcmp(file_write_option,"w")==0)))
            {
                vobjectPtr->setLock(); // set root lock
                vobjectPtr->setWorker();
                vobjectPtr->resetLastOpFalse();
                vobjectPtr->putObject();
            }
        }
        else
        {
            cout << "<Error getting thread in GET.VOBJECT.GRAPH:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET.VOBJECT.GRAPH:>\n";
    }
    break;
case 3:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {

```



```

threadPtr = (THREAD *)OC_lookup(operator_name);
if (threadPtr)
{
    V_OBJECT *vobjectPtr;
    vobjectPtr = threadPtr->version(atoi(versionstr));
    vobject_locktime = vobjectPtr->getLockTime();
    if (vobject_locktime>0) // prevent checkout
    {
        if (strcmp(file_write_option,"w")==0) // change "w" to "r"
        {
            cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                << " locked by : " << vobjectPtr->getWorker()
                << " Resetting write option to read-only>\n";
            strcpy(file_write_option,"r");
        }
        cerr << "<Caution: " << vobjectPtr->getNodeName()
            << " is locked.> \n"
            << "Date Locked: " << ctime(&vobject_locktime)
            << "operator files checked out in read-only mode\n";
    }
    else
        cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
    if ((strcmp(file_write_option,"w")==0) ||
        (strcmp(file_write_option,"w")==0))
    {
        vobjectPtr->setLock(); // set root lock
        vobjectPtr->setWorker();
        vobjectPtr->resetLastOpFalse();
        vobjectPtr->putObject();
    }
    COMPONENTPtr = vobjectPtr->getCOMPONENT();
    COMPONENTPtr->getGRAPHfile(file_write_option);
}
else
{
    cout << "<Error getting thread in GET_VOBJECT_GRAPH:>\n";
}
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_GRAPH>";
}
break;
default:
    cout << "<ERROR: invalid number args for get vobject GRAPH>\n";
}
}

void get_vobject_impfile_func(int number_arguments, char *proto_name,
                             char *operator_name, char *file_write_option, char *versionstr)

```

```

{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);

    switch (number_arguments)
    {
    case 2:
        prototypePtr = (PROTOTYPE*)OC_Lookup(prototype_name);
        if (prototypePtr)
        {
            threadPtr = (THREAD *)OC_Lookup(operator_name);
            if (threadPtr)
            {
                V_OBJECT *vobjectPtr;
                vobjectPtr = threadPtr->current();
                vobjectLocktime = vobjectPtr->getLockTime();
                if (vobjectLocktime>0) // prevent checkout
                {
                    if (strcmp(file_write_option,"w")==0) // change "w" to "r"
                    {
                        cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                            << " locked by : " << vobjectPtr->getWorker()
                            << " Resetting write option to read-only>\n";
                        strcpy(file_write_option,"r");
                    }
                    cerr << "<Caution: " << vobjectPtr->getNodeName()
                        << " is locked.> \n" << "Date Locked: "
                        << ctime(&vobjectLocktime)
                        << "operator files checked out in read-only mode\n";
                }
            }
            else
            {
                cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
                COMPONENTPtr = vobjectPtr->getCOMPONENT();
                if ((COMPONENTPtr->getIMPfile(file_write_option)) &&
                    ((strcmp(file_write_option,"w")==0) ||
                     (strcmp(file_write_option,"W")==0)))
                {
                    vobjectPtr->setLock(); // set root lock
                    vobjectPtr->setWorker();
                    vobjectPtr->resetLastOpFalse();
                    vobjectPtr->putObject();
                }
            }
        }
        else
        {
            cout << "<Error getting thread in GET.VOBJECT.IMP:>\n";
        }
    }
    else
    {

```

```

        cout << "<Error getting Prototype in GET_VOBJECT_IMP:>\n";
    }
    break;
case 3:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr->version(atoi(versionstr));
            vobjectPtr->getLockTime();
            if (vobjectPtr->locktime>0) // prevent checkout
            {
                if (strcmp(file_write_option,"w")==0) // change "w" to "r"
                {
                    cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                        << " locked by : " << vobjectPtr->getWorker()
                        << " Resetting write option to read-only>\n";
                    strcpy(file_write_option,"r");
                }
                cerr << "<Caution: " << vobjectPtr->getNodeName()
                    << " is locked.> \n"
                    << "Date Locked: " << ctime(&vobjectPtr->locktime)
                    << "operator files checked out in read-only mode\n";
            }
        }
        else
        {
            cerr << "MODULENAME ---> " << vobjectPtr->getNodeName() << "\n";
            if ((strcmp(file_write_option,"w")==0) ||
                (strcmp(file_write_option,"r")==0))
            {
                vobjectPtr->setLock(); // set root lock
                vobjectPtr->setWorker();
                vobjectPtr->resetLastOpFalse();
                vobjectPtr->putObject();
            }
            COMPONENTPtr = vobjectPtr->getCOMPONENT();
            COMPONENTPtr->getIMPfile(file_write_option);
        }
        else
        {
            cout << "<Error getting thread in GET_VOBJECT_IMP:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET_VOBJECT_IMP>";
    }
    break;
default:

```

```

        cout << "<ERROR: invalid number args for get vobject IMP>\n";
    }
}

void get_vobject_specfile_func(int number_arguments, char *proto_name,
                               char *operator_name, char *file_write_option, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr;
                    vobjectPtr = threadPtr->current();
                    vobject_locktime = vobjectPtr->getLockTime();
                    if (vobject_locktime > 0) // prevent checkout
                    {
                        if (strcmp(file_write_option, "w") == 0) // change "w" to "r"
                        {
                            cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                                << " locked by : " << vobjectPtr->getWorker()
                                << " Resetting write option to read-only>\n";
                            strcpy(file_write_option, "r");
                        }
                        else
                        {
                            cerr << "<Caution: " << vobjectPtr->getNodeName()
                                << " is locked.> \n" << "Date Locked: "
                                << ctime(&vobject_locktime)
                                << "operator files checked out in read-only mode\n";
                        }
                    }
                }
                else
                {
                    cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
                    COMPONENTPtr = vobjectPtr->getCOMPONENT();
                    if ((COMPONENTPtr->getSPECfile(file_write_option)) &&
                        ((strcmp(file_write_option, "w") == 0) ||
                         (strcmp(file_write_option, "r") == 0)))
                    {
                        vobjectPtr->setLock(); // set root lock
                        vobjectPtr->setWorker();
                        vobjectPtr->resetLastOpFalse();
                        vobjectPtr->putObject();
                    }
                }
            }
        }
    }
}

```

```

        else
        {
            cout << "<Error getting thread in GET_VOBJECT_SPEC:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in GET_VOBJECT_SPEC:>\n";
    }
    break;
case 3:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr->version(atoi(versionstr));
            vobjectPtr->getLockTime();
            if (vobjectPtr->locktime>0) // lock was set..prevent "w" checkout
            {
                if (strcmp(file_write_option,"w")==0) // if attempting "w" - change to "r"
                {
                    cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                        << " locked by : " << vobjectPtr->getWorker()
                        << " Resetting write option to read-only>\n";
                    strcpy(file_write_option,"r");
                }
                else
                {
                    cerr << "<Caution: " << vobjectPtr->getNodeName()
                        << " is locked.> \n" << "Date Locked: "
                        << ctime(&vobjectPtr->locktime)
                        << "operator files checked out in read-only mode\n";
                }
            }
            else
            {
                cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
                COMPONENTPtr = vobjectPtr->getCOMPONENT();
                if ((COMPONENTPtr->getSPECfile(file_write_option)) &&
                    ((strcmp(file_write_option,"W")==0) ||
                     (strcmp(file_write_option,"w")==0)))
                {
                    vobjectPtr->setLock(); // set root lock
                    vobjectPtr->setWorker();
                    vobjectPtr->resetLastOpFalse();
                    vobjectPtr->putObject();
                }
            }
        }
    }
    else
    {
        cout << "<Error getting thread in GET_VOBJECT_SPEC:>\n";
    }
}

```

```

    }
  }
  else
  {
    cout << "<Error getting Prototype in GET_VOBJECT_SPEC>";
  }
  break;
default:
  cout << "<ERROR: invalid number args for get vobject SPEC>\n";
}
}

```

```

void get_vobject_sourcefile_func(int number_arguments, char *proto_name,
                                char *operator_name, char *file_write_option, char *versionstr)
{
  char *prototype_name = new char [strlen(proto_name)+5];
  strcpy(prototype_name, proto_name);
  strcat(prototype_name, PROTOTYPE_EXT);

  switch (number_arguments)
  {
    case 2:
      prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
      if (prototypePtr)
      {
        threadPtr = (THREAD*)OC_lookup(operator_name);
        if (threadPtr)
        {
          V_OBJECT *vobjectPtr;
          vobjectPtr = threadPtr->current();
          vobject_locktime = vobjectPtr->getLockTime();
          if (vobject_locktime>0) // prevent checkout
          {
            if (strcmp(file_write_option, "w")==0) // change "w" to "r"
            {
              cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                << " locked by : " << vobjectPtr->getWorker()
                << " Resetting write option to read-only>\n";
              strcpy(file_write_option, "r");
            }
          }
          else
          {
            cerr << "<Caution: " << vobjectPtr->getNodeName()
              << " is locked.> \n" << "Date Locked: "
              << ctime(&vobject_locktime)
              << "operator files checked out in read-only mode\n";
          }
        }
      }
      else
      {
        cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
        COMPONENTPtr = vobjectPtr->getCOMPONENT();
        if ((COMPONENTPtr->getSOURCEfile(file_write_option)) &&
            ((strcmp(file_write_option, "w")==0) ||

```

```

        (strcmp(file_write_option,"w")==0)))
    {
        vobjectPtr → setLock();
        vobjectPtr → setWorker();
        vobjectPtr → resetLastOpFalse();
        vobjectPtr → putObject();
    }
}
else
{
    cout << "<Error getting thread in GET_VOBJECT_SOURCE:>\n";
}
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_SOURCE:>\n";
}
}
break;
case 3:
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypePtr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr → version(atoi(versionstr));
            vobjectLocktime = vobjectPtr → getLockTime();
            if (vobjectLocktime > 0) // prevent checkout
            {
                if (strcmp(file_write_option,"w")==0) // change "w" to "r"
                {
                    cerr << "<ERROR: Module " << vobjectPtr → getNodeName()
                        << " locked by : " << vobjectPtr → getWorker()
                        << " Resetting write option to read-only>\n";
                    strcpy(file_write_option,"r");
                }
            }
            else
            {
                cerr << "<Caution: " << vobjectPtr → getNodeName()
                    << " is locked.> \n" << "Date Locked: "
                    << ctime(&vobjectLocktime)
                    << "operator files checked out in read-only mode\n";
            }
        }
        else
        {
            cerr << "NODENAME ---> " << vobjectPtr → getNodeName() << "\n";
            COMPONENTPtr = vobjectPtr → getCOMPONENT();
            if ((COMPONENTPtr → getSourcefile(file_write_option)) &&
                ((strcmp(file_write_option,"w")==0) ||
                 (strcmp(file_write_option,"w")==0)))
            {
                vobjectPtr → setLock(); // set root lock
            }
        }
    }
}

```

```

        vobjectPtr → setWorker();
        vobjectPtr → resetLastOpFalse();
        vobjectPtr → putObject();
    }
}
else
{
    cout << "<Error getting thread in GET_VOBJECT_SOURCE:>\n";
}
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_SOURCE>";
}
break;
default:
    cout << "<ERROR: invalid number args for get vobject SOURCE>\n";
}
}

void dump_vobject_files_func(int number_arguments, char *proto_name,
                             char *operator_name, char *file_write_option, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);
    Boolean file_operation_successful = FALSE;

    switch (number_arguments)
    {
    case 2:
        prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
        if (prototypePtr)
        {
            threadPtr = (THREAD *)OC_lookup(operator_name);
            if (threadPtr)
            {
                V_OBJECT *vobjectPtr;
                vobjectPtr = threadPtr → current();
                vobject_locktime = vobjectPtr → getLockTime();
                if (vobject_locktime > 0)
                    // lock was set..prevent "w" checkout
                {
                    if (strcmp(file_write_option, "w") == 0)
                        // if attempting "W" - change to "r"
                    {
                        cerr << "<ERROR: Module "
                            << vobjectPtr → getNodeName() << " locked by : "
                            << vobjectPtr → getWorker()
                            << " Resetting write option to read-only>\n";
                        strcpy(file_write_option, "r");
                    }
                }
            }
        }
    }
}

```



```

    }
    else
        cerr << "<Caution: " << vobjectPtr->getNodeName()
            << " is locked.> \n" << "Date Locked: "
            << ctime(&vobjectPtr->locktime)
            << "operator files checked out in read-only mode\n";
    }
    else
        cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
    file_operation_successful =
        vobjectPtr->checkoutCOMPONENTNode(file_write_option);
    if ((file_operation_successful) &&
        ((strcmp(file_write_option, "w")==0) ||
         (strcmp(file_write_option, "w")==0)))
    {
        vobjectPtr->setLock(); // set root lock
        vobjectPtr->setWorker();
        vobjectPtr->resetLastOpFalse();
        vobjectPtr->putObject();
    }
    if (!file_operation_successful)
        cerr << "<Error checking out " << vobjectPtr->getName() :
    }
    else
    {
        cout << "<Error getting thread in GET_VOBJECT_FILES:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_FILES:>\n";
}
break;
case 3:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr;
            vobjectPtr = threadPtr->version(atoi(versionstr));
            vobjectPtr->getLockTime();
            if (vobjectPtr->locktime>0) // prevent checkout
            {
                if (strcmp(file_write_option, "w")==0) // change "w" to "r"
                {
                    cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                        << " locked by : " << vobjectPtr->getWorker()
                        << " Resetting write option to read-only>\n";
                }
            }
        }
    }
}

```

```

        strcpy(file_write_option,"r");
    }
    else
        cerr << "<Caution:  " << vobjectPtr->getNodeName()
        << " is locked.> \n" << "Date Locked:  "
        << ctime(&vobject_locktime)
        << "operator files checked out in read-only mode\n";
    }
    else
        cerr << "NODENAME ---> " << vobjectPtr->getNodeName() << "\n";
    COMPONENTPtr = vobjectPtr->getCOMPONENT();
    file_operation_successful =
        vobjectPtr->checkoutCOMPONENTNode(file_write_option);
    if ((file_operation_successful) &&
        ((strcmp(file_write_option,"w")==0) ||
         (strcmp(file_write_option,"w")==0)))
    {
        vobjectPtr->setLock(); // set root lock
        vobjectPtr->setWorker();
        vobjectPtr->resetLastOpFalse();
        vobjectPtr->putObject();
    }

    if (!file_operation_successful)
        cerr << "<Error checking out " << vobjectPtr->getName() ;
    }
    else
    {
        cout << "<Error getting thread in GET_VOBJECT_FILES:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in GET_VOBJECT_FILES>";
}
break;
default:
    cout << "<ERROR: invalid number args for get vobject FILES>\n";
}
}

void dump_vobject_tree_func(int number_arguments, char *proto_name,
                           char *operator_name, char *file_write_option, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name,proto_name);
    strcat(prototype_name,PROTOTYPE_EXT);
    Boolean file_operation_successful = FALSE;

    switch (number_arguments)
    {

```

case 3:

```

prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
if (prototypeptr)
{
    threadPtr = (THREAD *)OC_lookup(operator_name);
    if (threadPtr)
    {
        V_OBJECT *vobjectPtr = threadPtr->current();
        vobject_locktime = vobjectPtr->getLockTime();
        if (vobject_locktime>0) // prevent checkout
        {
            if (strcmp(file_write_option,"w")==0) // change "w" to "r"
            {
                cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                    << " locked by : " << vobjectPtr->getWorker()
                    << " Resetting write option to read-only>\n";
                strcpy(file_write_option,"r");
            }
            cerr << "<Caution: " << vobjectPtr->getNodeName()
                << " is locked.> \n" << "Date Locked: "
                << ctime(&vobject_locktime)
                << "Subtree checked out in read-only mode\n";
        }
        else
            cerr << "NODENAME ---> " << vobjectPtr->getNodeName()
                << "\nVersion: " << vobjectPtr->getVersionNumber() << "\n\n";
        file_operation_successful =
            vobjectPtr -> checkoutCOMPONENTNode(file_write_option);
        if ((file_operation_successful) &&
            ((strcmp(file_write_option,"w")==0) ||
             (strcmp(file_write_option,"W")==0)))
        {
            vobjectPtr -> setLock(); // set root lock
            vobjectPtr -> setWorker();
            vobjectPtr -> resetLastOpFalse();
            vobjectPtr->putObject();
        }
        if (file_operation_successful)
            vobjectPtr -> dumpSubtree(file_write_option);
        // dump rest of tree
        else
            cerr << "<Error checking out " << vobjectPtr ->getName()
                << " Aborting dump_vobject_tree.func>\n";
    }
    else
    {
        cout << "<Error getting thread in DUMP_VOBJECT_TREE:>\n";
    }
}
else
{

```

```

        cout << "<Error getting Prototype in DUMP_VOBJECT_TREE:>\n";
    }
    break;
case 4:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr = threadPtr->version(atoi(versionstr));
            vobject_locktime = vobjectPtr->getLockTime();
            if (vobject_locktime>0) // prevent checkout
            {
                if (strcmp(file_write_option,"w")==0) // change "w" to "r"
                {
                    cerr << "<ERROR: Module " << vobjectPtr->getNodeName()
                        << " locked by : " << vobjectPtr->getWorker()
                        << " Resetting write option to read-only>\n";
                    strcpy(file_write_option,"r");
                }
                else
                {
                    cerr << "<Caution: " << vobjectPtr->getNodeName()
                        << " is locked.> \n" << "Date Locked: "
                        << ctime(&vobject_locktime)
                        << "Subtree checked out in read-only mode\n";
                }
            }
            else
            {
                cerr << "NODENAME ---> " << vobjectPtr->getNodeName()
                    << "\nVersion: " << vobjectPtr->getVersionNumber() << "\n\n";
                file_operation_successful =
                    vobjectPtr->checkoutCOMPONENTNode(file_write_option);
                if ((file_operation_successful) &&
                    ((strcmp(file_write_option,"w")==0) ||
                     (strcmp(file_write_option,"W")==0)))
                {
                    vobjectPtr->setLock(); // set root lock
                    vobjectPtr->setWorker();
                    vobjectPtr->resetLastOpFalse();
                    vobjectPtr->putObject();
                }
            }
            if (file_operation_successful)
                vobjectPtr->dumpSubtree(file_write_option);
            // dump rest of tree
            else
            {
                cerr << "<Error checking out " << vobjectPtr->getName()
                    << " Aborting dump.vobject_tree.func>\n";
            }
        }
    }
    else
    {

```

```

        cout << "<Error getting thread in DUMP_VOBJECT.TREE:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in DUMP_VOBJECT.TREE>";
}
break;
default:
    cout << "<ERROR: invalid number args for get vobject TREE FILES>\n";
}
}

```

```

void long_list_operators_func(int number_arguments, char *proto_name,
                             char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr = threadPtr->current();
                    vobjectPtr->longlistOperatorNames();
                }
            }
            else
            {
                cout << "<Error getting thread in long list operators:>\n";
            }
        }
        else
        {
            cout << "<Error getting Prototype in long list operators:>\n";
        }
        break;
        case 3:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr = threadPtr->version(atoi(versionstr));
                    vobjectPtr->longlistOperatorNames();
                }
            }
        }
    }
}

```

```

    }
    else
    {
        cout << "<Error getting thread in long list operators:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in long list operators>";
}
break;
default:
    cout << "<ERROR: invalid number args for long list operators>\n";
}
}

```

```

void long_list_children_func(int number_arguments, char *proto_name,
                             char *operator_name, char *versionstr)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr = threadPtr->current();
                    vobjectPtr->listChildren();
                }
            }
            else
            {
                cout << "<Error getting thread in LONG_LIST_CHILDREN:>\n";
            }
        }
        else
        {
            cout << "<Error getting Prototype in LONG_LIST_CHILDREN:>\n";
        }
        break;
        case 3:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)

```

```

        {
            V_OBJECT *vobjectPtr = threadPtr->version(atoi(versionstr));
            vobjectPtr->listChildren();
        }
        else
        {
            cout << "<Error getting thread in LONG_LIST_CHILDREN:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in LONG_LIST_CHILDREN>";
    }
    break;
default:
    cout << "<ERROR: invalid number args for long list children >\n";
}
}

```

```

void long_list_parents_func(int number_arguments, char *proto_name,
                           char *operator_name, char *versionstr)

```

```

{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);

    switch (number_arguments)
    {
        case 2:
            prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
            if (prototypeptr)
            {
                threadPtr = (THREAD *)OC_lookup(operator_name);
                if (threadPtr)
                {
                    V_OBJECT *vobjectPtr = threadPtr->current();
                    V_OBJECT *parentPtr = vobjectPtr->getParent();
                    if (parentPtr)
                    {
                        V_OBJECT *grandparent = parentPtr->getParent();
                        if (grandparent)
                            grandparent->listChildren();
                        else
                            cout << " <\nRoot Node: "
                                << parentPtr->getNodeName() << ">\n";
                    }
                }
                else
                    cout << "<Already at Root V_OBJECT>\n";
            }
        else
        {

```

```

        cout << "<Error getting thread in LONG_LIST_CHILDREN:>\n";
    }
}
else
{
    cout << "<Error getting Prototype in LONG_LIST_CHILDREN:>\n";
}
break;
case 3:
    prototypePtr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypePtr)
    {
        threadPtr = (THREAD *)OC_lookup(operator_name);
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr = threadPtr->version(atoi(versionstr));
            V_OBJECT *parentPtr = vobjectPtr->getParent();
            if (parentPtr)
            {
                V_OBJECT *grandparent = parentPtr->getParent();
                if (grandparent)
                    grandparent->listChildren();
                else
                    cout << "<Can't list Parent/siblings of Root V_Object>\n";
            }
            else
                cout << "<Already at Root V_OBJECT>\n";
        }
        else
        {
            cout << "<Error getting thread in LONG_LIST_CHILDREN:>\n";
        }
    }
    else
    {
        cout << "<Error getting Prototype in LONG_LIST_CHILDREN>";
    }

    break;
default:
    cout << "<ERROR: invalid number args for long list children >\n";
}
}

void add_vobject_and_subtree_func(int number_arguments, char *proto_name,
                                char *operator_name)
{
    char *prototype_name = new char [strlen(proto_name)+5];
    strcpy(prototype_name, proto_name);
    strcat(prototype_name, PROTOTYPE_EXT);
}

```



```

switch (number_arguments)
{
case 2:
    prototypeptr = (PROTOTYPE*)OC_lookup(prototype_name);
    if (prototypeptr)
    {
        DIRECTORY *capsdirectory;
        capsdirectory = new DIRECTORY();
        capsdirectory->read_directory(operator_name);
        capsdirectory->updatetimestamp();
        TREENODE_linkedlist operatorList = capsdirectory->getOperatorList();
        TREENODE *rootnode = capsdirectory->find_treenode(operator_name);
        TREENODE *tree_root = new TREENODE(rootnode,NULL);
        TREE *workingtree = new TREE(tree_root, operator_name);
        workingtree->build_tree(tree_root,operatorList);
        cerr << "CHECKIN--> " << operator_name << "\n";
        threadPtr = (THREAD *)OC_lookup(operator_name);
        V_OBJECT *new_parent = (V_OBJECT *)0;
        if (threadPtr)
        {
            V_OBJECT *vobjectPtr = threadPtr->current();
            new_parent = vobjectPtr->getParent();
        }
        V_OBJECT *new_root = tree_root->checkin_node(new_parent);
        if (!new_root)
        {
            cerr << "<Error: Could not establish new_root in"
                << "add_vobject_and_subtree_func. Aborting.>\n";
            break;
        }
        new_root->setNodeName(tree_root->getname());
        tree_root->checkin_subtree(new_root);
    }
    else
    {
        cout << "<Error getting Prototype in ADD_VOBJECT_Subtree:>\n";
    }
    break;
default:
    cout << "<ERROR: invalid number args for add vobject & SUBTREE>\n";
}
}

```

APPENDIX E

SETUP FILES

A. ONTOS HEADERS/MAKEFILES PREPARATION NOTES

Only the ReferenceMacros.h ONTOS specific header file is included in this appendix due to its unique nature. It is a file built after the ONTOS system was released and was not included with the standard beta package.

The makefiles are included as an aid for the user in developing ONTOS applications.

B. PRINTING NOTES

This code was prepared for printout using the c++2latex code generator. This generator parses the ASCII text input files and places latex commands where directed. This makes the code layout more readable and highlights the important data structures and key words within the C++ code. The latex output was then converted to postscript format with the dviops program.

C. MAINTENANCE

This code is maintained at the Naval Postgraduate School Computer Science department.

```
// File Header -----
//.....:
//.Filename.....: ReferenceMacros.h
//.SCCS ID.....: 1.3
//.Release No....: 1
//.Date.....: 9/16/91
//.Compiler.....: Glockenspiel C++ 2.1
//.....:
// End header comments -----

/*
*****
*
* Disclaimer:
*
* This header file is a special header developed entirely by
* Ontologic, Inc.
* Copyright (c) 1990, Ontologic, Inc. USA All rights reserved.
*
* It's purpose is to handle Ontos References within the user code of
* the Design Database System and thus is included as an integral portion
* of the Design Database code.
*****
*/

#include <Reference.h>

// Note that the compiler must know that SpecType is an Entity*,
// e.g., by having seen the appropriate class definition.

// FOR Macro form use this define and continuation characters.

#define TypeCheckReference( SpecReference, SuperReference, SpecType )\
class SpecReference : public SuperReference {\
public:\
    SpecReference() {\}\
    SpecReference(SpecType* referent, Entity* context)\
    : (referent, context) {\}\
    SpecReference(SpecType* referent, StorageManager* context)\
    : (referent, context) {\}\
    SpecReference(SpecType* referent)\
    : (referent) {\}\
    void Reset(Entity* referent, Entity* context)\
    {\
        SuperReference::Reset(referent, context);\
    }\
    void Reset(Entity* referent, StorageManager* context)\
    {\

```

```

SuperReference::Reset(referent, context);\
}\
void Reset(Entity* referent)\
{\
SuperReference::Reset(referent);\
}\
void Reset(SpecReference& otherRef, \
StorageManager* oldContext, \
StorageManager* currentContext)\
{\
SuperReference::Reset(otherRef, oldContext, currentContext);\
}\
void Reset(SpecReference& otherRef, \
Entity* oldContext, \
Entity* currentContext)\
{\
SuperReference::Reset(otherRef, oldContext, currentContext);\
}\
void Init(SpecType* referent, Entity* context)\
{\
SuperReference::Init(referent, context);\
}\
void Init(SpecType* referent, StorageManager* context)\
{\
SuperReference::Init(referent, context);\
}\
void Init(SpecType* referent)\
{\
SuperReference::Init(referent);\
}\
void Init(SpecReference& otherRef, \
StorageManager* oldContext, \
StorageManager* currentContext)\
{\
SuperReference::Init(otherRef, oldContext, currentContext);\
}\
void Init(SpecReference& otherRef, \
Entity* oldContext, \
Entity* currentContext)\
{\
SuperReference::Init(otherRef, oldContext, currentContext);\
}\
SpecType* Binding(Entity* context, LockType lock=DefaultLock)\
{\
return (SpecType*) SuperReference::Binding(context, lock);\
}\
SpecType* Binding(StorageManager* context, \
LockType lock=DefaultLock)\
{\
return (SpecType*) SuperReference::Binding(context, lock);\
}\

```

```

SpecType* activeBinding(Entity* context)\
{\
return (SpecType*) SuperReference::activeBinding(context);\
}\
SpecType* activeBinding(StorageManager* context)\
{\
return (SpecType*) SuperReference::activeBinding(context);\
}\
} /* invocation supplies final ';' */

```

```

**
*****
**
** Ontologic, Inc.
** Copyright (c) 1990, Ontologic, Inc. USA All rights reserved.
**
*****
**

```

file: makefile

include make-macros

```

OBJECTS = main.o component.o versioned_object.o thread.o text_object.o \
prototype.o configuration.o evaluation.o directory.o \
tree.o treenode.o queue.o nodesupport.o protfunc.o \
conffunc.o vobjectfunc.o
CLASSIFY_HDRS = thread.h text_object.h prototype.h configuration.h \
component.h versioned_object.h
NONCLASSIFY_HDRS = evaluation.h directory.h tree.h treenode.h \
queue.h nodesupport.h protfunc.h \
conffunc.h vobjectfunc.h ddbdefines.h tracer.h
HEADERS = $(CLASSIFY_HDRS) $(NONCLASSIFY_HDRS)

```

```

#-----
# Objectfile/Headerfile dependencies:
# Persistent classes need their headers, and the application
# needs all headers.
#-----

```

```

tree.o: tree.h
nodesupport.o: nodesupport.h
protfunc.o: protfunc.h
evaluation.o: ddbdefines.h
component.o: component.h ddbdefines.h
thread.o: thread.h ddbdefines.h
versioned_object.o: versioned_object.h
text_object.o: text_object.h ddbdefines.h
configuration.o: configuration.h
prototype.o: prototype.h
main.o: $(HEADERS)

```

include make-targets

```
##
#####
##
## Ontologic, Inc.
## Copyright (c) 1990, Ontologic, Inc. USA All rights reserved.
##
#####
##

# file: make-macros

#-----
# Cplus sets up function bindings and then calls CC, but
# doesn't always produce the best warnings and error-messages
# So use CC when debugging warnings and errors, and use
# cplus to prepare for linking
# You can choose a compiler on the UNIX command line; just
# specify COMPILER, e.g., 'make COMPILER=CC'
#-----
COMPILER = $(ONTOS_BIN)/cplus
#COMPILER = CC

# We have versions for SUN and GLOCK
#WHICH_COMPILER = CC_SUN
WHICH_COMPILER = CC_GLOCK

#-----
# Flags for classify: See the System Reference

# By choosing EXTENSION=+X (perhaps on the command line) one
# enables instance-iteration
#
# By choosing VERBOSE=+v (perhaps on the command line) one
# can see output from 'classify'
#
#-----
VERBOSE =
#VERBOSE = +v
EXTENSION =
#EXTENSION = +X

ONTOS_DIR= /usr/local
ONTOS_BIN= $(ONTOS_DIR)/bin

INCLUDE = -I$(ONTOS_DIR)/include/ONTOS -I$(ONTOS_DIR)/include/hxx -I/usr/include
LIBRARY = -L$(ONTOS_DIR)/lib -IONTOS
```

```
KERNEL_DB = $(ONTOS_DIR)/ontos/db/OntosSchema
REGISTER_FLAG = $(THE_DB_DIR)/alreadyRegistered
```

```
SERVER_HOST = sun51
```

```
THE_DB_DIR = $(HOME)/ontos
MY_NAME = ddb53
```

```
first_target: again
```

```
##
##*****
##
## Ontologic, Inc.
## Copyright (c) 1990, Ontologic, Inc. USA All rights reserved.
##
##*****
##
```

```
# file: make-targets
```

```
#=====
#=====
# The first part of this file establishes relevant information.
#=====
#=====
```

```
#-----
# Extra flags for the compiler/linker. -g means include debug info.
# -Bstatic forces the loader to use static libraries, which may
# help debugging.
#-----
```

```
CFLAGS = -g
LFLAGS = -g -Bstatic
```

```
#-----
# How to make a .o when its .cxx has been touched.
#-----
```

```
.SUFFIXES: .cxx
.cxx.o:
@echo '... Compiling $*.cxx with $(COMPILER)'
@if [ "$(COMPILER)" = "CC" ]; \
then $(COMPILER) -c -D$(WHICH_COMPILER) $(CFLAGS) \
$(INCLUDE) $*.cxx; \
rm -f $*.o; \
else \
$(COMPILER) -c -D$(WHICH_COMPILER) $(CFLAGS) \
$(INCLUDE) $*.cxx; \
fi
```

```
#=====
```

```

=====
# The second part of this file uses the above information to
# let you 'make' the various goals.
=====
#-----
# Recompile/relink/reload-schemata if necessary
#-----
again: main loadTypes
test: again
main $(MY_NAME)

#-----
# Register, compile, link, load-schemata
#-----
first: register main loadTypes

#-----
# Link/compile the program, if necessary
#-----
main: $(OBJECTS)
@echo ... Linking main
@$(COMPILER) $(LFLAGS) -o main -QUIET \
$(OBJECTS) $(LIBRARY))

#-----
# Establish logical-to-physical mapping for database fragment
#-----
register:
@if [ -f $(REGISTER_FLAG) ]; then \
echo You are already registered!; \
exit 1; \
else touch $(REGISTER_FLAG); \
fi
#cp $(KERNEL_DB) $(THE_DB_DIR)/$(MY_NAME)
#chmod u+w $(THE_DB_DIR)/$(MY_NAME)
#@echo ... Running DBATool to register
#@$(ONTOS_BIN)/DBATool -e\
# register kernel $(MY_NAME)_kern on $(SERVER_HOST)\
# at $(THE_DB_DIR)/$(MY_NAME)
#@$(ONTOS_BIN)/DBATool -e\
# add area $(MY_NAME)_kern to $(MY_NAME)
#-----
# Add two more areas to the logical database, which initially
# contained only the kernel-area.
#
# We also illustrate the process of determining which area is
# primary.
#-----
#@$(ONTOS_BIN)/DBATool -e\
create area $(MY_NAME)_A1 on $(SERVER_HOST)\

```



```

at $(THE_DB_DIR)/$(MY_NAME)_area1
@$(ONTOS_BIN)/DBATool -e\
create area $(MY_NAME)_A2 on $(SERVER_HOST)\
at $(THE_DB_DIR)/$(MY_NAME)_area2
@$(ONTOS_BIN)/DBATool -e\
add area $(MY_NAME)_A1 to $(MY_NAME)
@$(ONTOS_BIN)/DBATool -e\
add area $(MY_NAME)_A2 to $(MY_NAME)
@$(ONTOS_BIN)/DBATool -e\
set db $(MY_NAME) primary $(MY_NAME)_kern
@$(ONTOS_BIN)/DBATool -e\
show db $(MY_NAME)
@$(ONTOS_BIN)/DBATool -e\
show db
#-----
# Make backup copies of the areas, so that we can always start
# fresh. We can't simply copy the master version of OntosSchema
# since (a) we can register a kernel only once and (b) when
# we register the kernel, OCServer changes an area-mapping
# section in the header of the area.
#-----
@echo
@echo Waiting for the above changes to take hold before
@echo making backup copies.
sleep 20
-mkdir $(THE_DB_DIR)/backup
cp $(THE_DB_DIR)/$(MY_NAME)_Kernel $(THE_DB_DIR)/backup
cp $(THE_DB_DIR)/$(MY_NAME)_area? $(THE_DB_DIR)/backup

#-----
# Refresh your Directory in the kernel database, and prime
# schema for reloading.
#-----
freshDB:
@rm -f loadTypes
@rm -f $(THE_DB_DIR)/JRN*
@cp $(THE_DB_DIR)/backup/* $(THE_DB_DIR)
chmod ugo+rw $(THE_DB_DIR)/$(MY_NAME)

#-----
# Load schemata, i.e., information about user-defined types
# into the database. Since we don't support migration, load
# is performed only on a fresh database
#
# +D indicates logical database.
#-----
loadTypes: $(CLASSIFY_HDRS)
@make freshDB
@make loadTypes2
@touch loadTypes

```

LIST OF REFERENCES

- [Ref. 1] Booch, G., *Software Engineering with ADA*, 2d ed., Benjamin/ Cummings, 1987.
- [Ref. 2] Tanik, M. M. & Yeh, R. T., "Rapid Prototyping in Software Development", *Computer*, v. 22, n. 5, pp. 9-11, May 1989.
- [Ref. 3] Cummings, M.A., *The Development of User Interface Tools for the Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, December, 1990.
- [Ref. 4] Luqi, *Software Evolution Through Rapid Prototyping*, IEEE Computer, v. 22, n. 5, pp. 13-25, May 1989.
- [Ref. 5] McDowell, J. K., *A Reusable Component Retrieval System for Prototyping*, Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1991.
- [Ref. 6] Bayramoglu, S., *The Design and Implementation of an Expander for the Hierarchical Real-Time Constraints of Computer-Aided Prototyping System (CAPS)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1991.
- [Ref. 7] Levine, J., *An Efficient Heuristic Scheduler for Hard Real-time Systems*, Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1991.
- [Ref. 8] Douglas, B. S., *A Conceptual Level Design of a Design Database for the Computer-Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989.
- [Ref. 9] Berzins, V. A., and Luqi, *Software Engineering with Abstraction*, Addison-Wesley Publishing Company, Inc., 1991.
- [Ref. 10] Berstein, P. A., "Database System Support for Software Engineering -- An Extended Abstract", IEEE 9th International Conference on Software, pp. 166-178, 1987.
- [Ref. 11] Luqi, *A Graph Model for Software Evolution*, IEEE Transactions on Software Evolution, v. 16, n. 8, pp. 917-927, August 1990.
- [Ref. 12] Loomis, M. E. S., "More on Transactions", *Journal of Object-Oriented Programming*, v. 3, n. 5, pp. 63-67, January 1990.

- [Ref. 13] Kim, W., "Architectural Issues in Object-Oriented Databases", *Journal of Object-Oriented Programming*, v. 2 n.6, pp. 29-38, March/April 1990.
- [Ref. 14] Rajiv, G. and Others, "An Object-Oriented VLSI CAD Framework A Case Study in Rapid Prototyping", *Computer*, v.22, n. 5, pp. 28-37, May 1989.
- [Ref. 15] Yourdon, E., *Modern Structured Analysis*, Yourdon Press, 1989.
- [Ref. 16] Barnes, P. , *Graphical User Interface for the CAPS Design Database*, Software Research Project, Naval Postgraduate School, 1991.
- [Ref. 17] Nestor, *Toward a Persistent Database*, Technical Memorandum, Software Engineering Institute, Carnegie-Mellon Institute, SEI-86-TM-8, pp 1-19DDB.
- [Ref. 18] Mullin, M., *Object Oriented Program Design*, Addison Wesley, 1989.
- [Ref. 19] Luqi, *Computer Aided Software Prototyping*, IEEE Computer, pp. 111-112, September 1991.

BIBLIOGRAPHY

Elmasri, R. & Navathe, S. B., *Fundamentals of Database Systems*, Benjamin/Cummings Publishing Company, 1989.

Fisher, A.S., *Using Software Development Tools*, John Wiley & Son, Inc., 1988.

Lippman, S. B., *C++ Primer*, Addison-Wesley Publishing Company, 1990.

Luqi, Ketabchi, *A Computer Aided Prototyping System*, IEEE Transactions on Software Evolution, March 1988.

McMenamin, S. M. & Palmer, J. F. , *Essential Systems Analysis*, Yourdon Press, April 1984.

Miller, *The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information*, The Psychological Review, v. 63, n. 2, pp. 81-97, March, 1956.

Ontologic Inc., *ONTOS Release 2.0 Product Description*, Burlington, MA.

Ontologic Inc., *ONTOS Object Database Documentation Release 1.5*, Burlington, MA .

Nelson, M. L., *Object-Oriented Database Management Systems*, Technical Report NPS52-90-025, Naval Postgraduate School, Monterey, California, May 1990.

Page-Jones, M., *The Practical guide to Structured Systems Design*, Yourdon Press, 1980.

Rumbaugh, J. and Others, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

Stroustrup, B., *The C++ Programming Language*, Addison Wesley, 1986.

Van Wyk, C. J., *Data Structures and C Programs*, Addison Wesley, 1988.

Ward, P. T., *Systems Development without Pain*, Yourdon Press, 1984.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
3. Computer Science Department Code CS Naval Postgraduate School Monterey, CA 93943	2
4. Office of the Assistant Secretary of the Navy Research Development and Acquisition Department of the Navy Attn: Mr. Gerald A. Cann Washington, DC 20380-1000	1
5. Office of the Chief of Naval Operations OP-094 Department of the Navy Attn: VADM J. O. Tuttle, USN Washington, DC 20301-3040	1
6. Director of Defense Information Office of the Assistant Secretary of Defense (Command, Control, Communications, & Intelligence) Attn: Mr. Paul Strassmann Washington, DC 20301-0208	1
7. Center for Naval Analysis 4401 Ford Avenue Alexandria, VA 22302-0268	1

8. Director of Research Administration Attn: Prof. Howard Code 08Hk Naval Postgraduate School Monterey, CA 93943	1
9. Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	1
10. Prof. Luqi, Code CSLq Computer Science Department Naval Postgraduate School Monterey, CA 93943	10
11. Chief of Naval Research 800 N. Quincy Street Arlington, VA 22217	1
12. Director, Ada Joint Program Office OUSDRE (R&AT) Room 3E114, The Pentagon Attn: Dr. John P. Solomond Washington, DC 20301-0208	1
13. Carnegie Mellon University Software Engineering Institute Attn: Dr. Dan Berry Pittsburgh, PA 15260	1
14. Office of Naval Technology (ONT) Code 227 Attn: Dr. Elizabeth Wald 800 N. Quincy St. Arlington, VA 22217-5000	1
15. Defense Advanced Research Projects Agency (DARPA) Integrated Strategic Technology Office (ISTO) Attn: Dr. B. Boehm 1400 Wilson Boulevard Arlington, VA 22209-2308	1

16. Defense Advanced Research Projects Agency (DARPA) 1
ISTO
1400 Wilson Boulevard
Attn: LCol Eric Mattala
Arlington, VA 2209-2308
17. Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, VA 2209-2308
18. Attn: Dr. Charles Harland 1
Computer Science
Department of the Air Force
Bolling Air Force Base
Washington, DC 20332-6448
19. Chief of Naval Operations 1
Attn: Dr. R. M. Carroll (OP-01B2)
Washington, DC 20350
20. Dr. Amiram Yehudai 1
Tel Aviv University
School of Mathematical Sciences
Department of Computer Science
Tel Aviv, Israel 69978
21. Dr. Robert M. Balzer 1
USC-Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina del Ray, CA 90292-6695
22. Dr. Ted Lewis 1
OR State University
Computer Science Department
Corvallis, OR 97331
23. International Software Systems Inc. 1
12710 Research Boulevard, Suite 301
Attn: Dr. R. T. Yeh
Austin, TX 78759

24. Kestrel Institute Attn: Dr. C. Green 1801 Page Mill Road Palo Alto, CA 94304	1
25. National Science Foundation Division of Computer and Computation Research Attn: K. C. Tai Washington, DC 20550	1
26. Commander Space and Naval Warfare Systems Command SPAWAR 3212 Department of the Navy Attn: Cdr M. Romeo Washington, DC 20363-5100	1
27. Naval Ocean Systems Center Attn: Linwood Sutton, Code 423 San Diego, CA 92152-5000	1
28. Office of Naval Research Computer Science Division, Code 1133 Attn: Dr. Gary Koob 800 N. Quincy Street Arlington, VA 22217-5000	1
29. Commander, Naval Sea Systems Command (PMS-4123H) Attn: William Wilder Washington, DC 20380-1000	1
30. New Jersey Institute of Technology Computer Science Department Attn: Dr. Peter Ng Newark, NJ 07102	1
31. Office of Naval Research Computer Science Division, Code 1133 Attn: Dr. A. M. Van Tilborg 800 N. Quincy Street Arlington, VA 22217-5000	1

32. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. R. Wachter
800 N. Quincy Street
Arlington, VA 22217-5000
33. OR Graduate Center 1
Portland (Beaverton)
Attn: Dr. R. Kieburtz
Portland, OR 97005
34. Santa Clara University 1
Department of Electrical Engineering and
Computer Science
Attn: Dr. M. Ketabchi
Santa Clara, CA 95053
35. Software Group, MCC 1
9430 Research Boulevard
Attn: Dr. L. Belady
Austin, TX 78759
36. University of CA at Berkeley 1
Department of Electrical Engineering and
Computer Science
Computer Science Division
Attn: Dr. C.V. Ramamoorthy
Berkeley, CA 90024
37. University of CA at Irvine 1
Department of Computer and Information Science
Attn: Dr. Nancy Leveson
Irvine, CA 92717
38. Chief of Naval Operations 1
Attn: Dr. Earl Chavis (OP-16T)
Washington, DC 20350
39. Office of the Chief of Naval Operations 1
Attn: Dr. John Davis (OP-094H)
Washington, DC 20350-2000

40. University of Illinois 1
Department of Computer Science
Attn: Dr. Jane W. S. Liu
Urbana Champaign, IL 61801
41. University of MD 1
College of Business Management
Tydings Hall, Room 0137
Attn: Dr. Alan Hevner
College Park, MD 20742
42. University of MD 1
Computer Science Department
Attn: Dr. N. Roussapoulos
College Park, MD 20742
43. University of Massachusetts 1
Department of Computer and Information Science
Attn: Dr. John A. Stankovic
Amherst, MA 01003
44. University of Pittsburgh 1
Department of Computer Science
Attn: Dr. Alfs Berztiss
Pittsburgh, PA 15260
45. University of TX at Austin 1
Computer Science Department
Attn: Dr. Al Mok
Austin, TX 78712
46. Commander, Naval Surface Warfare Center, 1
Code U-33
Attn: Dr. Philip Hwang
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000
47. Attn: George Sumiall 1
US Army Headquarters
CECOM
AMSEL-RD-SE-AST-SE
Fort Monmouth, NJ 07703-5000

48. Attn: Joel Trimble 1
1211 South Fern Street, C107
Arlington, VA 22202
49. United States Laboratory Command 1
Army Research Office
Attn: Dr. David Hislop
P. O. Box 12211
Research Triangle Park, NC 27709-2211
50. George Mason University 1
Computer Science Department
Attn: Dr. David Rine
Fairfax, VA 22030-4444
51. Hewlett Packard Research Laboratory 1
Mail Stop 321
1501 Page Mill Road
Attn: Dr. Martin Griss
Palo Alto, CA 94304
52. Carnegie Mellon University 1
SEI
Attn: Dr. Mario Barbacci
Pittsburgh, PA 15213
53. Persistent Data Systems 1
75 W. Chapel Ridge Road
Attn: Dr. John Nester
Pittsburgh, PA 15238
54. Sun Microsystems Inc. 1
MS MTV100-01
Silicon Valley Government District
1842 N. Shoreline Boulevard
Attn: Vice President c/o Don Chandler
Mountain View, CA 94043
55. Commandant of the Marine Corps 1
Ada Joint Program Representative
Code CCI
Attn: Capt Gerald Depasquale
Washington, DC 20301

56. Commandant of the Marine Corps Code TE-06 Washington, DC 20301	1
57. Commandant of the Marine Corps Code CCT-60 Attn: Ltcol L. Machabee Washington, DC 20301	1
58. Commanding General Marine Corps Research, Development, and Acquisition Command Attn: LtCol Garry W. Lewis Quantico, VA 22134	1
59. Ontologic, Inc. Three Burlington Woods Attn: Mr. Gregory Harris Burlington, MA 01803	1