REPORT

AD-A251 316

Public reporting burden for this collection
and maintaining the data needed, and revi
suggestions for reducing this burden, to W
22202-4302, and to the Office of Informati

E 4 | Form Approved OPM No.

(2)

e time for reviewing instructions, searching existing data sources gathering
den estimate or any other aspect of this collection of information, including
s and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA
ton, DC 20503.

1. AGENCY USE (Leave		3. REPORT TYPE AND DATES
		Final: 17 Dec 1991

4. TITLE AND
Validation Summary Report: TeleSoft, TeleGen2 Ada Compilation System for VAX to 80960, Version 4.1, MicroVAX 3800 under VAX/VMS Version V5.4(Host) Intel EXV 960 MC-MIL (i960 XA) (bare target), 911213I1.11235

5. FUNDING

6.
IABG-AVF
Ottobrunn, Federal Republic of Germany

7. PERFORMING ORGANIZATION NAME(S) AND
IABG-AVF, Industrieanlagen-Betriebsgeselschaft
Dept. SZT/ Einsteinstrasse 20
D-8012 Ottobrunn
FEDERAL REPUBLIC OF GERMANY

8. PERFORMING ORGANIZATION
IABG-VSR 100

9. SPONSORING/MONITORING AGENCY NAME(S) AND
Ada Joint Program Office
United States Department of Defense
Pentagon, Rm 3E114
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY

11. SUPPLEMENTARY

12a. DISTRIBUTION/AVAILABILITY
Approved for public release; distribution unlimited.

12b. DISTRIBUTION

13. (Maximum 200
TeleSoft, TeleGen2 Ada Compilation System for VAX to 80960, Version 4.1, MicroVAX 3800 under VAX/VMS Version V5.4(Host) Intel EXV 960 MC-MIL (i960 XA) (bare target), ACVC 1.11

14. SUBJECT
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A,

15. NUMBER OF

16. PRICE

17. SECURITY CLASSIFICATION	18. SECURITY	19. SECURITY CLASSIFICATION	20. LIMITATION OF
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFIED	

NSN

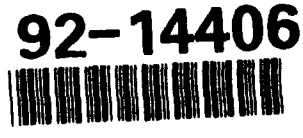
Standard Form 298, (Rev. 2-89)
Prescribed by ANSI Std.

AVF Control Number: IABG-VSR 100
17 December 1991

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 911213I1.11235
TeleSoft
TeleGen2™ Ada Compilation System
for VAX to 80960, Version 4.1
MicroVAX 3800 under VAX/VMS Version V5.4 =>
Intel EXV 960 MC-MIL (i960 XA) (bare target)

== based on TEMPLATE Version 91-05-08 ==

Prepared By:
IABG mbH, Abt. ITE
Einsteinstr. 20
W-8012 Ottobrunn
Germany



Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 91-12-13.

Compiler Name and Version:	**TeleGen2™ Ada Compilation System for VAX to 80960, Version 4.1**
Host Computer System:	**MicroVAX 3800 under VAX/VMS Version V5.4**
Target Computer System:	**Intel EXV 960 MC-MIL (i960 XA) running the Ada RTS interface of Hughes O.S. (bare target)**

See Section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate #911213I1.11235 is awarded to TeleSoft. This certificate expires on 01 June 1993.

This report has been reviewed and is approved.

IABG, Abt. ITE
Michael Tonndorf
Einsteinstr. 20
W-8012 Ottobrunn
Germany

for

Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

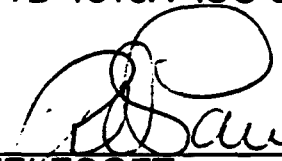
Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense

DECLARATION OF CONFORMANCE

Customer:	TeleSoft 5959 Cornerstone Court West San Diego CA USA 92121
Ada Validation Facility:	IABG, Dept. ITE W-8012 Ottobrunn Germany
ACVC Version:	1.11
Ada Implementation:	
Ada Compiler Name and Version:	TeleGen2™ Ada Compilation System for VAX to 80960, Version 4.1
Host Computer System:	MicroVAX 3800 (under VAX/VMS Version V5.4)
Target Computer System:	Intel EXV 960 MC-MIL (i960 XA) *running* the Ada RTS interface of Hughes O. S. (bare target)

Customer's Declaration

I, the undersigned, declare that TeleSoft has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A ISO 8652-1987 in the implementation listed above.



Date: 12/10/91

TELESOFT
Raymond A. Parra
V. P., General Counsel

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Computer and Software Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311-1772

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.

Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 02 August 1991.

E28005C	B28006C	C32203A	C34006D	C35508I	C35508J
C35508M	C35508N	C35702A	C35702B	B41308B	C43004A
C45114A	C45346A	C45612A	C45612B	C45612C	C45651A
C46022A	B49008A	B49008B	A74006A	C74308A	B83022B
B83022H	B83025B	B83025D	B83026B	C83026A	C83041A
B85001L	C86001F	C94021A	C97116A	C98003B	BA2011A
CB7001A	CB7001B	CB7004A	CC1223A	BC1226A	CC1226B
BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A	CD2A21E
CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A	CD2B15C
BD3006A	BD4008A	CD4022A	CD4022D	CD4024B	CD4024C
CD4024D	CD4031A	CD4051D	CD5111A	CD7004C	ED7005D
CD7005E	AD7006A	CD7006E	AD7201A	AD7201E	CD7204B
AD7206A	BD8002A	BD8004C	CD9005A	CD9005B	CDA201E
CE2107I	CE2117A	CE2117B	CE2119B	CE2205B	CE2405A
CE3111C	CE3116A	CE3118A	CE3411B	CE3412B	CE3607B
CE3607C	CE3607D	CE3812A	CE3814A	CE3902B	

2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

The following 159 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113O..Y (11 tests)	C35705O..Y (11 tests)
C35706O..Y (11 tests)	C35707O..Y (11 tests)
C35708O..Y (11 tests)	C35802O..Z (12 tests)
C45241O..Y (11 tests)	C45321O..Y (11 tests)
C45421O..Y (11 tests)	C45521O..Z (12 tests)
C45524O..Z (12 tests)	C45621O..Z (12 tests)
C45641O..Y (11 tests)	C46012O..Z (12 tests)

The following 20 tests check for the predefined type LONG_INTEGER; for this implementation, there is no such type:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45613C	C45614C	C45631C	C45632C	B52004D
C55B07A	B55B09C	B86001W	C86006C	CD7101F

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT; for this implementation, there is no such type.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

C45624A..B (2 tests) check that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types; for this implementation, MACHINE_OVERFLOWS is TRUE.

B86001Y checks for a predefined fixed-point type other than DURATION; for this implementation, there is no such type.

CA2009C and CA2009F check whether a generic unit can be instantiated before its body (and any of its subunits) is compiled; this implementation creates a dependence on generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic unit bodies makes the instantiating units obsolete. (See section 2.3.)

LA3004A..B, EA3004C..D, and CA3004E..F (6 tests) check pragma INLINE for procedures and functions; this implementation does not support pragma INLINE.

CD1009C uses a representation clause specifying a non-default size for a floating-point type; this implementation does not support such sizes.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use representation clauses specifying non-default sizes for access types; this implementation does not support such sizes.

The following 264 tests check operations on sequential, text, and direct access files; this implementation does not support external files:

CE2102A. C (3)	CE2102G..H (2)	CE2102K	CE2102N..Y (12)
CE2103C..D (2)	CE2104A..D (4)	CE2105A..B (2)	CE2106A..B (2)
CE2107A..H (8)	CE2107L	CE2108A..H (8)	CE2109A..C (3)
CE2110A..D (4)	CE2111A..I (9)	CE2115A..B (2)	CE2120A..B (2)
CE2201A..C (3)	EE2201D..E (2)	CE2201F..N (9)	CE2203A
CE2204A..D (4)	CE2205A	CE2206A	CE2208B
CE2401A..C (3)	EE2401D	CE2401E..F (2)	EE2401G
CE2401H..L (5)	CE2403A	CE2404A..B (2)	CE2405B
CE2406A	CE2407A..B (2)	CE2408A..B (2)	CE2409A..B (2)
CE2410A..B (2)	CE2411A	CE3102A..C (3)	CE3102F..H (3)
CE3102J..K (2)	CE3103A	CE3104A..C (3)	CE3106A..B (2)
CE3107B	CE3108A..B (2)	CE3109A	CE3110A
CE3111A..B (2)	CE3111D..E (2)	CE3112A..D (4)	CE3114A..B (2)
CE3115A	CE3119A	EE3203A	EE3204A
CE3207A	CE3208A	CE3301A	EE3301B
CE3302A	CE3304A	CE3305A	CE3401A
CE3402A	EE3402B	CE3402C..D (2)	CE3403A..C (3)
CE3403E..F (2)	CE3404B..D (3)	CE3405A	EE3405B
CE3405C..D (2)	CE3406A..D (4)	CE3407A..C (3)	CE3408A..C (3)
CE3409A	CE3409C..E (3)	EE3409F	CE3410A
CE3410C..E (3)	EE3410F	CE3411A	CE3411C
CE3412A	EE3412C	CE3413A..C (3)	CE3414A
CE3602A..D (4)	CE3603A	CE3604A..B (2)	CE3605A..E (5)
CE3606A..B (2)	CE3704A..F (6)	CE3704M..O (3)	CE3705A..E (5)
CE3706D	CE3706F..G (2)	CE3804A..P (16)	CE3805A..B (2)
CE3806A..B (2)	CE3806D..E (2)	CE3806G..H (2)	CE3904A..B (2)
CE3905A..C (3)	CE3905L	CE3906A..C (3)	CE3906E..F (2)

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 15 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B71001Q	BA1001A	BA2001C	BA2001E	BA3006A
BA3006B	BA3007B	BA3008A	BA3008B	BA3013A

C52008B was graded passed by Test Modification as directed by the AVO. This test uses a record type with discriminants with defaults; this test also has array components whose length depends on the values of some discriminants of type INTEGER. On elaboration of the type declaration, this implementation raises NUMERIC_ERROR as it attempts to calculate the maximum possible size for objects of the type. The AVO ruled that this behavior was acceptable, and that the test should be modified to constrain the subtype of the discriminants. Line 16 was modified to create a constrained subtype of INTEGER, and discriminant specifications in lines 17 and 25 were modified to use that subtype; these lines are given below:

```
16   SUBTYPE SUBINT IS INTEGER RANGE -128 .. 127;
17   TYPE REC1(D1,D2 : SUBINT) IS
25   TYPE REC2(D1,D2,D3,D4 : SUBINT := 0) IS
```

CA2009C and CA2009F were graded inapplicable by Evaluation Modification as directed by the AVO. These tests contain instantiations of a generic unit prior to the compilation of that unit's body; as allowed by AI-00408 and AI-00506, the compilation of the generic unit bodies makes the compilation unit that contains the instantiations obsolete.

BC3204C and BC3205D were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete--no errors are detected. The processing of these tests was modified by re-compiling the obsolete units; all intended errors were then detected by the compiler.

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For technical and sales information about this Ada implementation, contact:

TeleSoft
5959 Cornerstone Court West
San Diego, CA 92121-3731, USA
(619) 457-2700

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system -- if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

a)	Total Number of Applicable Tests	3603	
b)	Total Number of Withdrawn Tests	95	
c)	Processed Inapplicable Tests	49	
d)	Non-Processed I/O Tests	264	
e)	Non-Processed Floating-Point Precision Tests	159	
f)	Total Number of Inapplicable Tests	472	(c+d+e)
g)	Total Number of Tests for ACVC 1.11	4170	(a+b+f)

3.3 TEST EXECUTION

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded onto the host computer via DECNET.

After the test files were loaded onto the host computer, the full set of tests except 264 I/O Tests and 159 Floating-Point Precision Tests were processed by the Ada implementation.

Test output, compiler and linker listings, and job logs were captured on a magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing are given on the two next pages, which were supplied by the customer.

Compiler Option Information

B Tests and E Tests:

tsada/E960 ada/monitor/enable/list/virtual_space=10000 <test_name>

option	description
tsada	invoke Ada compilation system
E960	specify the 80960 target
ada	compile the test
monitor	verbose mode, output pass info during compilation
enable	enable traceback for unexpected errors
list	generate compilation listing
virtual_space	memory allocation control
<test_name>	name of Ada source file to be compiled

All Execution Tests except E Tests:

tsada/E960 ada/monitor/virtual_space=10000/nolist <test_name>

option	description
tsada	invoke Ada compilation system
E960	specify the 80960 target
ada	compile the test
monitor	verbose mode, output pass info during compilation
virtual_space	memory allocation control
nolist	do not generate compilation listing
<test_name>	name of Ada source file to be compiled

BIND:

tsada/E960 bind/monitor/virtual_space=10000 <main_unit>

option	description
tsada	invoke Ada compilation system
E960	specify the 80960 target
bind	bind the main unit
monitor	verbose mode, output pass info during compilation
virtual_space	memory allocation control
<main_unit>	name of main compilation unit

LINK:

tsada/E960 link/monitor/options=mcmil_main.opt/enable/virtual=10000 <main_unit>

option	description
tsada	invoke Ada compilation system
E960	specify the 80960 target
link	link the test
monitor	verbose mode, output pass info during compilation
map	generate link map listing
options	specify linker options file
enable	enable traceback for unexpected errors
virtual_space	memory allocation control
<main_unit>	name of main compilation unit

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$MAX_IN_LEN	200 -- Value of V
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"
$MAX_STRING_LITERAL	"CCCCCCC10CCCCCCCC20CCCCCCCC30CCCCCCCC40 CCCCCCCC50CCCCCCCC60CCCCCCCC70CCCCCCCC80 CCCCCCCC90CCCCCCC100CCCCCCC110CCCCCCC120 CCCCCCC130CCCCCCC140CCCCCCC150CCCCCCC160 CCCCCCC170CCCCCCC180CCCCCCC190CCCCCC199"

The following table lists all of the other macro parameters and their respective values.

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_646
$DEFAULT_MEM_SIZE	2147483647
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	TELESOFT_ADA
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	INTERRUPT1
$ENTRY_ADDRESS1	INTERRUPT2
$ENTRY_ADDRESS2	INTERRUPT3
$FIELD_LAST	1000
$FILE_TERMINATOR	ASCII.EOT
$FIXED_NAME	NO_SUCH_TYPE
$FLOAT_NAME	LONG_LONG_FLOAT
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	100_000.0
$GREATER_THAN_DURATION_BASE_LAST	131_073.0
$GREATER_THAN_FLOAT_BASE_LAST	3.9E+39
$GREATER_THAN_FLOAT_SAFE_LARGE	1.0E38
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE	0.0
$HIGH_PRIORITY	31

```
$ILLEGAL_EXTERNAL_FILE_NAME1
                        BADCHAR*^/%

$ILLEGAL_EXTERNAL_FILE_NAME2
                        /NONAME/DIRECTORY

$INAPPROPRIATE_LINE_LENGTH
                        -1

$INAPPROPRIATE_PAGE_LENGTH
                        -1

$INCLUDE_PRAGMA1        PRAGMA INCLUDE ("A28006D1.ADA")

$INCLUDE_PRAGMA2        PRAGMA INCLUDE ("B28006D1.ADA")

$INTEGER_FIRST          -2147483648

$INTEGER_LAST           2147483647

$INTEGER_LAST_PLUS_1    2147483648

$INTERFACE_LANGUAGE     C

$LESS_THAN_DURATION     -100_000.0

$LESS_THAN_DURATION_BASE_FIRST
                        -131_073.0

$LINE_TERMINATOR        ASCII.CR

$LOW_PRIORITY           0

$MACHINE_CODE_STATEMENT
                        MCI'(addi,g0,g0,g0);

$MACHINE_CODE_TYPE      INSTRUCTION;

$MANTISSA_DOC           31

$MAX_DIGITS             18

$MAX_INT                2147483647

$MAX_INT_PLUS_1         2147483648

$MIN_INT                -2147483648

$NAME                   SHORT_SHORT_INTEGER

$NAME_LIST              TELESOFT_ADA

$NEG_BASED_INT          16#FFFFFFFE#

$NEW_MEM_SIZE           2147483467
```

$NEW_SYS_NAME	TELESOFT_ADA
$PAGE_TERMINATOR	ASCII.FF
$RECORD_DEFINITION	record null; end record;
$RECORD_NAME	INSTRUCTION;
$TASK_SIZE	32
$TASK_STORAGE_SIZE	4096
$TICK	0.01
$VARIABLE_ADDRESS	ADDRESS1
$VARIABLE_ADDRESS1	ADDRESS2
$VARIABLE_ADDRESS2	ADDRESS3

APPENDIX B

COMPILATION SYSTEM AND LINKER OPTIONS

The compiler and linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

TELESOFT

80960 ACS Ada Compilation System for VAX/VMS Systems to Embedded 80960 Targets

Compiler Command Options

OPT-1941N-V1.1(VAX.E960) 02DEC91

Version 4.01.03

TeleSoft
5959 Cornerstone Court West
San Diego, CA 92121-9819
(619) 457-2700
(Contractor)

1 Compiler command options

1.1. ADA

[Compiler]

Introduction

After access to 80960 ACS has been established and a library has been created, you can invoke the Ada compiler via the ADA command. The general command format for compilation is

```
TSADA/E960 ADA[/<qualifier>[,...]] <file>
```

where

`<qualifier>` One of the qualifiers available for the compiler.

`<file>` One in a possible series of file specifications, separated by commas, indicating the unit(s) to be compiled. If /INPUT_LIST is used, <file> is interpreted as a file containing a list of files to be compiled. The default source file type is .ADA, and the default list file type is .LIS. A file name may be qualified with a location in standard VAX/VMS format. A source or input list file may reside on any directory in the system.

The default qualifier settings are designed to allow for the simplest and most convenient use of the compiler. For most applications, no additional qualifier setting need be specified. However, optional qualifiers are provided to perform special functions.

Qualifiers

/ABORT_COUNT=<n>
/ABORT_COUNT=999 (default)

The /ABORT_COUNT qualifier is an execution control qualifier for the compiler. It allows you to set the maximum number of errors the compiler can locate before it aborts. This qualifier can be used with any combination of compiler options. The minimum value is 1 and the default value is 999.

The compiler maintains separate counts of all syntactic errors, semantic errors, and warning messages issued during a compilation. If any of these counts becomes too great, you may want to abort the compilation. If the compiler does find errors, you can abort the compilation by entering control-Y (^Y) or you can wait until the checking is completed. The /ABORT_COUNT qualifier allows you to determine the number of errors that you believe is reasonable before the compiler aborts.

/ASSEMBLY_CODE[=<file>]
/NOASSEMBLY_CODE (default)

With the /ASSEMBLY_CODE qualifier, you can obtain an assembly listing of compiler generated code for a unit or a collection. You can use this qualifier with the compiler, the binder and the optimizer in the same way. The listing is similar to that produced with the /MACHINE_CODE qualifier, except that it will not contain location or offset information.

The file produced with the /ASSEMBLY_CODE qualifier is suitable as input to an assembler. In contrast, the file produced by /MACHINE_CODE is more suitable for human readability. The default file name is the same as it would be for /MACHINE_CODE, i.e., the unit name being compiled. /ASSEMBLY_CODE and /MACHINE_CODE are mutually exclusive.

/BIND[=<main>][<qualifier>]
/NOBIND (default)

This qualifier, when used with the compiler, enables you to bind a main program more efficiently by combining the binding process with the compilation process. This feature is especially useful when compiling single-unit test programs. The binding of more than one main program is not supported, though programs that contain multiple compilation units may be bound.

If you do not specify a main unit with the /BIND qualifier, the last unit to be compiled will be the unit that is bound as a main unit.

Default settings for the qualifier values were chosen for the simplest and most convenient use of the binder. For most applications, no additional qualifiers are required. Optional qualifiers are provided, as shown in the list that follows. For more information on these qualifiers, see the BIND command in this "Command Summary" chapter.

/LIBFILE	Specify a library other than the default LIBLST.ALB.
/TEMPLIB	Specify a library other than the default LIBLST.ALB.
/SHOW_TASK_EXCEPTIONS	Cause unhandled exceptions in tasks to be reported in the same manner as those that occur in the main program.
/TASK_STACK_SIZE	Set the default amount of stack to allocate from the Ada heap for each task.
/TIME_SLICE_QUANTUM	Specify the amount of time, in milliseconds, in which a task is allowed to execute before the run-time switches control to another ready task of equal priority.
/TRACEBACK	Set the depth of the run-time exception traceback report.

Note: The main program unit must be located in the working sublibrary. If this is not the case, reorder the sublibraries in the library file or use the library manager's MOVE or COPY commands to move or copy the unit to the working sublibrary.

If these conditions are met, you may proceed to bind and link the program.

/CONTEXT=<n>
/CONTEXT=1 (default)

When an error message is output, it is helpful to include the lines of the source program that surround the line containing the error. These lines provide a context for the error in the source program and help to clarify the nature of the error. The /CONTEXT qualifier controls the number of source lines that immediately precede and follow the error. The qualifier affects the error messages output on SYS$OUTPUT. The default setting for this qualifier is 1.

/COPY_SOURCE
/NOCOPY_SOURCE

This qualifier tells the compiler to take the source file and store it in the Ada library. When you need to retrieve your source file later, use the EXTRACT command.

/DEBUG
/NODEBUG (default)

To use the debugger, you must compile, or assemble and link program units using the /DEBUG qualifier. This ensures that source debugging information is properly stored for later use by the debugger. The /DEBUG qualifier causes the compiler to save High Form for debugging purposes. It also causes the compiler to generate Debugging Information (DI) for any unit that is to be used with the debugger.

Because this qualifier is positional, the debug information is generated only for the specified files. For example,

```
$ TSADA/E960 ADA A,B/DEBUG,C
```

compiles A.ADA and B.ADA without debugging information, and compiles C.ADA with debugging information. For information on positional qualifiers, see the "Overview" chapter in the *Overview and Command Summary*.

The default setting of this qualifier is /NODEBUG. The use of /DEBUG ensures that the High Form and debugger information for secondary units are not deleted. While the compilation time overhead generated by use of /DEBUG is minimal, retaining this optional information in the Ada Library increases the space overhead. To check if a compilation unit has been compiled using /DEBUG, use the SHOW/EXTENDED command for the unit.

/DIAGNOSTICS[=<file>]
/NODIAGNOSTICS (default)

If warnings or errors are encountered during compilation, this qualifier specifies that the compiler or binder is to produce a diagnostics file (with the file type .DIA). This file contains information that allows you to use the VAX Language Sensitive Editor (LSE) to quickly correct source errors. More information can be found in the LSE manual.

This qualifier is positional, so the location of the command and its parameters on the command line is important. Only diagnostics files for the specified files are generated.

/ENABLE_TRACEBACK
/NOENABLE_TRACEBACK (default)

In the unlikely event that you should receive an Unexpected Error Condition message, you should contact Customer Support. Customer Support may request that you provide additional information about the error condition by including the /ENABLE_TRACEBACK qualifier in the compiler invocation that fails. This qualifier allows the tool to display the exception traceback associated with the unexpected error condition. The information provided in the traceback will allow Customer Support to diagnose the problem more efficiently.

/GRAPH[=<file>]
/NOGRAPH (default)

The /GRAPH qualifier is used with the optimizer (/OPTIMIZE) while compiling to generate a textual representation call graph for the unit being compiled and optimized. This qualifier is positional, so the location of the command and its parameters on the command line is important. The specified parameters for /GRAPH are the graph files to be generated. For instance, the command

```
TSADA/E960 ADA/OPTIMIZE=ALL A,B/GRAPH, C
```

compiles A.ADA, then sends a graph to B.GRF, then compiles C.ADA. For information on positional qualifiers, see the "Overview" chapter in the *Overview and Command Summary*.

The default setting is /NOGRAPH. If you specify graph, the following file specifications can result:

`<file>`	A name you specify for the file to which the generated graph is sent.
`<unit>.GRF`	A name provided for the file by default, where <unit> is the name of the unit being compiled when optimizing during compilation.

If multiple units are being optimized, and call graphs are desired, /GRAPH should be used without specifying a file name. The graph for each unit will be located in a separate file named after the unit. If you specify a <file>, separate versions of the file will be created for each unit.

/INPUT_LIST
/NOINPUT_LIST (default)

As a default, the file list specified in the TSADA/E960 command is the list of specifications of the files containing the Ada units to be compiled. When you specify the /INPUT_LIST qualifier, the compiler assumes that the command parameters are specifications of files which contain lists of source files to be compiled. When this qualifier is used, the input files should contain the source file specifications, one per line. For example, to compile source files CALC_ARITH.ADA and CALC_MEM.ADA in the current default directory and file CALC_IO.ADA in directory [CIOSRC], you could first prepare a file CALC_COMPILE.LIS containing the following text:

```
CALC_ARITH
CALC_MEM
[CIOSRC]CALC_IO
```

The command:

```
$ TSADA/E960 ADA/INPUT_LIST CALC_COMPILE.LIS
```

would then cause these three files to be compiled in sequence.

In addition to the names of the source files, the input list may contain comments in Ada syntax, i.e., all text on a line including and following the comment marker "--" will be ignored.

/INPUT_LIST is a positional qualifier. The specified parameters are input lists, and all other parameters are source files. For example, the command line

```
$ TSADA/E960 ADA A,B/INPUT_LIST,C
```

compiles A.ADA first. It then compiles all files in B.LIS, and finally, it compiles C.ADA.

/INPUT_LIST has the same advantages and effects as a multi-file compilation specified on the command line. See the "Compiler" chapter in *User Guide, Part A* for a full description. /INPUT_LIST is useful when a long list of files to compile has been readily obtained from a directory listing or a SHOW library report.

When the /INPUT_LIST qualifier is used in conjunction with the default /UPDATE qualifier, the working sublibrary is updated after each unit that successfully compiles. If a unit in the list fails to compile due to errors, the sublibrary is still updated for all other successfully compiled units before and after it in the list. If /NOUPDATE is used, the sublibrary is not updated for any unit if one unit fails to compile. See the /UPDATE entry in this Appendix for more information.

If the /BIND qualifier is used with the /INPUT_LIST qualifier, the main program unit name may be given as the value of the /BIND qualifier to identify which unit in the list is the main program. If not specified, the last unit of the input list is assumed to be the main program.

/LIBFILE=<file>
/LIBFILE=LIBLST (default)

By default, the library file named LIBLST with a default type of .ALB is used by the 80960 ACS tool set to determine which set of sublibraries are to be referred to during the operation of the tool. This file must be present in the working directory. With the /LIBFILE qualifier, you can specify a library other than the default, LIBLST. The /TEMPLIB qualifier may also be used to create an alternative library. However, the /TEMPLIB and /LIBFILE qualifiers are mutually exclusive; only one or the other qualifier may be used at the same time.

When you specify the /LIBFILE qualifier, you indicate the file specification of an alternative library file that contains the list of sublibraries and optional comments. If you do not specify a file type with the file name, the system uses the file type .ALB.

For example, consider a library file named WORKLIB.ALB with the contents:

```
Name: MYWORK
Name: [CALCPROJ]CALCLIB
Name: TSADA$E960:[LIB]RTL_960.SUB
```

You could specify

```
$ TSADA/E960 ADA/LIBFILE=WORKLIB
```

As an alternative to using /LIBFILE, you may assign the library file specification to the logical name LIBLST. For example,

```
$ ASSIGN WORKLIB.ALB LIBLST
```

/LINK=<main>[<qualifier>]
/NOLINK (default)

This qualifier invokes the linker when you are compiling. Using only one command line, you can compile, bind and link a program. This feature can be very useful when compiling single-unit test programs. If you are using the ADA command to invoke the linker, the program must be compiled and bound as well as linked. Because you cannot bind more than one main program when you compile and bind in the same process, you also cannot link more than one main program. You can compile, bind and link programs that contain multiple compilation units.

Default settings for the /LINK qualifier values are similar to those used with the standalone invocation of the linker. For most applications, no additional qualifiers are required. Optional qualifiers are provided, as shown in the list that follows. For more information on these qualifiers, see the LINK command in this "Command Summary" chapter.

/BASE	Specify the starting location of the linked output.
/EXCLUDED	Show the excluded subprograms in the link map.
/FORMAT	Output a load module in one of these formats: TeleSoft EFORM, Motorola S-records, IEEE-695 object format, Microtec S-records, OASYS object format, or a user-adapted object module format.
/IMAGE	Include a memory image listing in the link map.
/LOAD_MODULE	Specify the VMS file name for the load module output created by the linker.
/LOCALS	Includes local symbols in the link map symbol listing.
/MAP	Requests and controls a link map listing.
/OFORM	Specifies that one output of the linker is to be linked OF.
/OPTIONS	Specifies that the linker is to process additional options obtained interactively, or from a linker options file.
/PAGE	Output a user-specified number of lines per listing page.
/SYMBOL_FILE	This qualifier produces a file that contains all of the global symbols used in the link. The default file name is <main>.SYM.,
/VIRARS_SIZE	Specifies the amount of space, in kilobytes, of buffer space to be allocated for the linker.
/WIDTH	Specifies the width of the lines in the listing file.

/LIST=<file>
/NOLIST (default)

The /LIST qualifier to the ADA compiler command produces a file containing a source listing with numbered lines and any error messages. This qualifier is positional, so the location of the qualifier and its parameters on the command line is important. A list will only be generated for the specified parameters. For information on positional qualifiers, see the "Overview" chapter in the *Overview and Command Summary*.

The compiler always outputs error messages to the device specified by SYS$OUTPUT. The /LIST qualifier causes the error listing to be incorporated in the source listing file as well. The default listing contains the errors intermixed with the source code.

You can provide a file specification to the /LIST qualifier which indicates the VMS file to receive the error output.

The default is /NOLIST. The /NOLIST qualifier will suppress error output to the listing file.

If the /LIST qualifier is used without a file specification, the error output is sent to the file named <file>.LIS, where <file> is the name of the source file being compiled. If you provide a file specification, the output is sent to the specified file instead of <file>.LIS. If your file specification does not include a file type, the system gives the file the type, .LIS.

If a file name is specified and multiple source files are being compiled, the listing for each file is output to a separate version of the file name specified.

/MACHINE_CODE[=<file>]
/NOMACHINE_CODE (default)

The /MACHINE_CODE qualifier allows you to obtain an assembly listing of the code that the compiler generates for a unit or a collection. The listing consists of assembly code intermixed with source code as comments. Note that the listing generated by this qualifier is independent of the source/error listing generated by the /LIST qualifier. The default for this qualifier is /NOMACHINE_CODE.

The listing output is sent to a file named <unit>_.S if the unit is a library unit, and <unit>.S if the unit is a secondary unit. <unit> is the name of the compilation unit that is being listed.

If multiple compilation units are being compiled and you have provided a file specification, the machine code listing for each compilation unit will be output to a different version of the same file name.

If the compilation unit name is longer than 39 characters, the name will be truncated at 39 characters. No listing will be generated if there are syntactic or semantic errors in the compilation.

/MONITOR
/NOMONITOR (default)

Normally, the only visible output produced by the 80960 ACS tool set during operation is error or warning messages. The /MONITOR qualifier enables the reporting of version numbers and messages that allow you to monitor the tool's progress. When you specify /MONITOR, the output is sent to standard output (SYS$OUTPUT).

/OBJECT (default)
/NOOBJECT

The /NOOBJECT qualifier instructs the compiler to perform syntactic and semantic analysis of the source program without generating object code. /NOOBJECT sets a default of /NOSQUEEZE to ensure that the High Form and Low Form are preserved for secondary units. The default setting is /OBJECT, which allows the generation of object code.

/OPTIMIZE[=<option>[,...]][<qualifier>]
/NOOPTIMIZE (default)

The /OPTIMIZE qualifier causes the compiler to invoke the optimizer to optimize the Low Form generated by the middle pass for the unit being compiled. The code generator takes the optimized Low Form as input and produces more efficient object code.

/NOOPTIMIZE is the default. The /NOOPTIMIZE qualifier gives the quickest compilation turnaround, but does not perform many code optimizations. This results in code that may run slower and be larger than normal. Code intended for use with the debugger must be compiled with /NOOPTIMIZE. You should be familiar with the information on the optimizer presented in *User Guide, Part A* before you use this qualifier.

This qualifier is positional, so the location of the qualifier and its parameters is important. /OPTIMIZE will only optimize the specified parameters. For example, the qualifier may appear on both on the verb that is changing the default and on one or more parameters. The verb qualifier indicates that all parameters will be optimized. The parameter qualifier indicates that Y is not to be optimized.

```
$ TSADA/E960 ADA/OPTIMIZE X,Y/NOOPTIMIZE,Z
```

The result is that X.ADA is compiled and optimized, followed by the file Y.ADA. The file Z.ADA is compiled, but not optimized.

There are two parameters that are used with /OPTIMIZE. These are:

`<option>` An optimizer option used on the command line. A list of these options would be separated by commas.

`<qualifier>` The qualifier /NOGRAPH or /GRAPH[=<file>] /NOGRAPH is the default value. /GRAPH generates a call graph for the unit being compiled. The default output file is <unit>.GRF when you do note specify a <file>. (See the "Optimizer" chapter in *User Guide, Part B* for more information on the generated graph.)

You can specify zero or more optimizer options to control optimization. These options are listed here briefly. For more information on these options, see the OPTIMIZE command in this chapter.

Table -1. Optimizer Options

Options	Defaults	Operation
`ALL | NONE | SAFE`	`SAFE`	Enable/disable certain optimizations, or permit only safe optimizations.
`[NO]AUTOINLINE`	`AUTOINLINE`	Controls automatic inlining.
`[NO]INLINE[:<file>]`	`INLINE`	Enables inline expansion of subprograms.
`[NO]PARALLEL`	`PARALLEL`	Subprograms may be called from parallel tasks.
`[NO]RECURSE`	`RECURSE`	Subprograms may be called recursively.

/SQUEEZE (default)
/NOSQUEEZE

When you compile an Ada program, the compiler stores two intermediate code representations of the program in the library. These code representations are known as High Form and Low Form. High Form must be retained for a library unit because it is required for the compilation of any units that reference it.

For example, a compiled package specification's High Form are used by the corresponding package body when it is compiled. However, intermediate forms of a secondary unit, such as a package body, may frequently be discarded after its compilation. Discarding this information results in a significant decrease in library size (typically 50 to 70 percent for multi-unit programs).

The /SQUEEZE qualifier can be used with the compiler or the optimizer (OPTIMIZE). Using the /SQUEEZE qualifier during compilation causes the intermediate forms to be discarded after compilation, if possible. /NOSQUEEZE causes the full intermediate forms to be saved in all cases.

Note: The optimizer (OPTIMIZE), and cross-referencer (XREF) programs require unsqueezed units. If you are going to use one of these programs, you must compile the units using /NOSQUEEZE.

The default for this qualifier is /SQUEEZE, with one exception. This is the /NOOBJECT qualifier which is commonly used when compiling units for collective optimization. In this case, the object code is not required, but unsqueezed units are. Thus, use of the /NOOBJECT qualifier also causes /NOSQUEEZE to be the default. In either of these cases, use of an explicit /SQUEEZE or /NOSQUEEZE qualifier overrides the default.

To verify whether or not a unit has been squeezed, use the SHOW/EXTENDED command for the unit as, as described in the "Library Manager" chapter in *User Guide, Part A*. A unit has not been squeezed if and only if the attributes High_Form and Low_Form appear in the listing for that unit.

/SUPPRESS[=(<option>[,...])]
/NOSUPPRESS (default)

The /SUPPRESS qualifier allows you to suppress selected run-time checks and/or source line references in generated object code.

The Ada language requires, as a default, a wide variety of run-time checks to ensure the validity of operations. For example, arithmetic overflow checks are required on all numeric operations, and range checks are required on all assignment statements that could result in an illegal value being assigned to a variable. While these checks are vital during development and an important asset of the language, they introduce a substantial overhead. This overhead may be prohibitive in time-critical applications. Thus, the Ada language provides a way to selectively suppress classes of checks via the Suppress pragma. However, use of the pragma requires modifications to the Ada source.

The /SUPPRESS qualifier provides a functional alternative to the Suppress pragma. /SUPPRESS allows you to suppress checks in the compiler invocation command without modifying the source code. The Suppress pragma is valid in any declarative region of a package and affects all nested regions. The /SUPPRESS qualifier is equivalent to adding pragma Suppress to the beginning of the declarative part of each compilation unit in a file.

The compiler also stores source line and subprogram name information by default in the object code. This information is used to display a source level traceback when an unhandled exception propagates to the outer level of a program. This information is also particularly valuable during development as it provides a direct indication of the source line at which the exception occurs and the subprogram calling chain that led to the line generating the exception.

The source line information introduces an overhead of 6 bytes for each line of source that causes code to be generated. Thus, a 1000-line package may have up to 6000 bytes of source information. For one compilation unit, the extra overhead (in bytes), is the total length of all subprogram names in the unit (including Middle Pass generated subprograms), plus the length of the compilation unit name. For certain space-critical applications, this extra space may be unacceptable and may be inhibited with the /SUPPRESS qualifier. When the source line information is inhibited, the traceback indicates the offset of the object code at which the exception occurs, instead of the source line number. When the subprogram name information is inhibited, the traceback indicates the offsets of the subprogram calls in the calling chain, instead of the subprogram names.

When you specify an <option>, it represents one of a possible list of options separated by commas. These options indicate the features to be suppressed. The default setting is /NOSUPPRESS.

The options and their actions are presented in the following table. The names of the options may be abbreviated as long as they remain unique within the set of options. All options except SOURCE_INFO and ALL function as if a corresponding Suppress pragma were present in the Ada source. The exception is that /SUPPRESS=(ELABORATION_CHECK) differs from pragma Suppress(Elaboration_Check). The switch suppresses elaboration checks made by other units on this unit. The pragma suppresses elaboration checks made on other units from this unit. The NAME_INFO option specifies that subprogram name information is to be suppressed in the object code. The SOURCE_INFO option specifies that source line information is to be suppressed in the object code. The ALL_CHECKS option suppresses all run-time checks listed in the table. The ALL option specifies that subprogram name information, source line information, and all run-time checks in the table are to be suppressed.

For example, the qualifier:

```
/SUPPRESS=(SOURCE,ELAB)  MY_FILE
```

inhibits the generation of source line information and elaboration checks in the object code of the units in file MY_FILE.

ALL	Suppress source line information and all run-time checks listed below.
NONE	Equivalent to /NOSUPPRESS
SOURCE_INFO	Suppress source line information in object code.
NAME_INFO	Suppress subprogram name information in object code.
ALL_CHECKS	Suppress all access checks, discriminant checks, division checks, elaboration checks, index checks, length checks, overflow checks, range checks, and storage checks.
ELABORATION_CHECK	Suppress all elaboration checks.
OVERFLOW_CHECK	Suppress all overflow checks.

/TEMPLIB=(<sublib>[,...])
/LIBFILE=LIBLST (default)

This qualifier allows you to define a temporary library consisting of a selection of sublibraries. The temporary library may be used for the duration of a single command. In all uses, the /TEMPLIB and /LIBFILE options are mutually exclusive; only one or the other qualifier may be used at the same time.

<sublib> The name of the sublibrary, optionally prefixed with the specification of the VMS directory in which it resides. If no directory is specified, the current default directory is assumed. Multiple sublibrary file specifications are separated by commas in a list.

Semantically, the argument string of this qualifier is the logical equivalent of a library file containing the listed sublibraries, one per line, in the order listed. Thus, we could list the sublibraries:

```
/TEMPLIB=(MYWORK,[CALCPROJ]CALCLIB,TSADA$E960:[LIB]RTL_E960.SUB)
```

/UPDATE (default)
/[NO]UPDATE

When multiple source files are being compiled, the /UPDATE qualifier instructs the compiler to update the library after each source file is compiled. The default setting is /UPDATE.

If /NOUPDATE is used, and an error occurs during compilation, the working sublibrary is not updated at all, for any unit, even for remaining units in the source file in error. All remaining source files will be compiled for syntactic and semantic errors only. The /NOUPDATE qualifier is advantageous to use when it is known that all the source files will compile without error and the user wishes to save the overhead time involved in updating the library for each source file.

/VIRTUAL_SPACE=<n>
/VIRTUAL_SPACE=10000 (default)

This qualifier specifies the number of 1kb pages that will be used in memory while the tool executes. Greater values will usually improve performance, but will result in more physical memory requirements.

1.2. BIND

[Binder]

Introduction

The object code files generated by the compiler are 80960 ACS-defined Object Form files stored in the Ada library. These files must be bound to create a linkable object. The binder program generates the code needed to elaborate the components in a consistent order. The binder is invoked when you use the BIND command.

The general command format of the bind step is

```
$ TSADA/E960 BIND[/<qualifier>,...] <main>
```

where

`<main>` The name of the unit to be used as the main program

`<qualifier>` A binder qualifier

Information on the qualifiers and parameters used to invoke the binder are listed below.

Qualifiers

/ASSEMBLY_CODE[=<file>]
/NOASSEMBLY_CODE (default)

With the /ASSEMBLY_CODE qualifier, you can obtain an assembly listing of compiler generated code for a unit or a collection. You can use this qualifier with the compiler, the binder and the optimizer in the same way. The listing is similar to that produced with the /MACHINE_CODE qualifier, except that it will not contain location or offset information.

The file produced with the /ASSEMBLY_CODE qualifier is suitable as input to an assembler. In contrast, the file produced by /MACHINE_CODE is more suitable for human readability. The default file name is the same as it would be for /MACHINE_CODE, i.e., the unit name being compiled. /ASSEMBLY_CODE and /MACHINE_CODE are mutually exclusive.

/ENABLE_TRACEBACK
/NOENABLE_TRACEBACK (default)

In the unlikely event that you should receive an Unexpected Error Condition message, you should contact Customer Support. Customer Support may request that you provide additional information about the error condition by including the /ENABLE_TRACEBACK qualifier in the compiler invocation that fails. This qualifier allows the tool to display the exception traceback associated with the unexpected error condition. The information provided in the traceback will allow Customer Support to diagnose the problem more efficiently.

/LIBFILE=<file>
/LIBFILE=LIBLST (default)

By default, the library file named LIBLST with a default type of .ALB is used by the 80960 ACS tool set to determine which set of sublibraries are to be referred to during the operation of the tool. This file must be present in the working directory. With the /LIBFILE qualifier, you can specify a library other than the default, LIBLST. The /TEMPLIB qualifier may also be used to create an alternative library. However, the /TEMPLIB and /LIBFILE qualifiers are mutually exclusive; only one or the other qualifier may be used at the same time.

When you specify the /LIBFILE qualifier, you indicate the file specification of an alternative library file that contains the list of sublibraries and optional comments. If you do not specify a file type with the file name, the system uses the file type .ALB.

For example, consider a library file named WORKLIB.ALB with the contents:

```
Name: MYWORK
Name: [CALCPROJ]CALCLIB
Name: TSADA$E960:[LIB]RTL_960.SUB
```

You could specify

```
$ TSADA/E960 BIND/LIBFILE=WORKLIB <main>
```

As an alternative to using /LIBFILE, you may assign the library file specification to the logical name LIBLST. For example,

```
$ ASSIGN WORKLIB.ALB LIBLST
```

/MACHINE_CODE[=<file>]
/NOMACHINE_CODE (default)

The /MACHINE_CODE qualifier allows you to obtain an assembly listing of the code that the compiler generates for a unit or a collection. The listing consists of assembly code intermixed with source code as comments. Note that the listing generated by this qualifier is independent of the source/error listing generated by the /LIST qualifier. The default for this qualifier is /NOMACHINE_CODE.

The listing output is sent to a file named <unit>_.S if the unit is a library unit, and <unit>.S if the unit is a secondary unit. <unit> is the name of the compilation unit that is being listed.

If multiple compilation units are being compiled and you have provided a file specification, the machine code listing for each compilation unit will be output to a different version of the same file name.

If the compilation unit name is longer than 39 characters, the name will be truncated at 39 characters. No listing will be generated if there are syntactic or semantic errors in the compilation.

/MONITOR
/NOMONITOR (default)

Normally, the only visible output produced by the 80960 ACS tool set during operation is error or warning messages. The /MONITOR qualifier enables the reporting of version numbers and messages that allow you to monitor the tool's progress. When you specify /MONITOR, the output is sent to standard output (SYS$OUTPUT).

/SHOW_TASK_EXCEPTIONS
/NOSHOW_TASK_EXCEPTIONS (default)

The /SHOW_TASK_EXCEPTIONS qualifier causes unhandled exceptions in tasks to be reported in the same manner as those that occur in the main program. If you use this qualifier with the ADA command, it is only valid if you also use the /BIND qualifier.

/TASK_STACK_SIZE=<n>
/TASK_STACK_SIZE=4096 (default)

The /TASK_STACK_SIZE qualifier sets the default amount of stack to allocate from the Ada heap for each task. The <n> you specify is the size of the task stack in bytes. If you use this qualifier with the ADA command, you must also use the /BIND qualifier.

/TEMPLIB=(<sublib>[,...])
/LIBFILE=LIBLST (default)

This qualifier allows you to define a temporary library consisting of a selection of sublibraries. The temporary library may be used for the duration of a single command. In all uses, the /TEMPLIB and /LIBFILE options are mutually exclusive; only one or the other qualifier may be used at the same time.

`<sublib>` The name of the sublibrary, optionally prefixed with the specification of the VMS directory in which it resides. If no directory is specified, the current default directory is assumed. Multiple sublibrary file specifications are separated by commas in a list.

Semantically, the argument string of this qualifier is the logical equivalent of a library file containing the listed sublibraries, one per line, in the order listed. Thus, we could list the sublibraries:

```
/TEMPLIB=(MYWORK,[CALCPROJ]CALCLIB,TSADA$E960:[LIB]RTL_E960.SUB)
```

The /TEMPLIB qualifier applies to both compilation and binding, so it need be specified only once and may appear in any order on the command line when you use the BIND command. The binder needs to have present every compilation unit referenced by the main program. If a unit is missing, the binder will report the error and will not be invoked. Therefore, you should be sure that the set of sublibraries specified by the /TEMPLIB qualifier contains all the units belonging to the main program.

/TIME_SLICE_QUANTUM=<n>
/TIME_SLICE_QUANTUM=0 (default)

This qualifier specifies the slice of time, in milliseconds, in which a task is allowed to execute before the run-time switches control to another ready task of equal priority. This timeslicing activity allows for periodic round-robin scheduling among equal-priority tasks. Timeslicing may or may not be implemented for a particular environment.

The default value for /TIME_SLICE_QUANTUM is 0 (i.e., timeslicing is disabled). No run-time overhead is incurred when timeslicing is disabled. This qualifier must be used with the /BIND qualifier when you are compiling with the ADA command.

/TRACEBACK
/NOTRACEBACK (default)

This qualifier sets the default of the run-time exception traceback report.

If you use this qualifier with the ADA command, you must also use the /BIND qualifier.

/VIRTUAL_SPACE=<n>
/VIRTUAL_SPACE=5000 (default)

This qualifier specifies the number of 1kb pages that will be used in memory while the tool executes. Greater values will usually improve performance, but will result in more physical memory requirements.

1.3. LINK

[Linker]

Introduction

The Ada Linker is a component of the 80960 ACS system that allows you to link compiled Ada programs in preparation for target execution. The linker resolves references within the Ada program, the bare target run-time support library, and any imported non-Ada object code. To support the development of embedded applications, the linker is designed to operate in a variety of modes and to handle many types of output format.

The linker links together OF modules to construct executable load modules. (See the "Linker" chapter in the *User Guide* for details). Optionally, the linker outputs symbol location information that is used by the debugger. The linker can also output information used by the profiler. All unused subprograms will be eliminated from the executable image.

The command syntax for the Ada Linker is:

```
$ TSADA/E960 LINK[/<qualifier>[,...]] [<unit>]
```

where

`<qualifier>` One of the command line qualifiers available for the linker.

`<unit>` An optional command line parameter indicating the name of the Ada compilation unit to be linked as a main program.

Note that the compilation unit must have been bound as a main program prior to linking. If you do not provide the unit name on the command line, then the unit is specified using the INPUT option in an options file.

Linker directives are communicated to the linker as qualifiers on the VMS command line or as options entered via an options file or SYS$INPUT. Command line qualifiers are useful for controlling options that you are likely to change often. The default qualifier settings are designed to allow for the simplest and most convenient use of the linker.

Command line qualifiers and parameters enable you to:

- Specify the name of the linked output file
- Control the generation and format of listing map files
- Specify an options file
- Specify the library file containing the components to be linked
- Control the output of debug symbol information for debugging
- Monitor the linking process

More complicated linker options, such as the specification of memory locations for specific portions of the code or data for a program, are input via options in a linker options file. Linker options may be used to:

- Specify the compilation units to be used as input to the Linker, the library search paths, and the usage of the input files
- Specify the name and format of the linked output file
- Control the generation and format of listing map files produced by the Linker.
- Specify the location of named memory regions and reserved memory regions in physical memory
- Specify the location of control sections in physical memory
- Define symbol values

Each of the LINK command qualifiers is described in detail in the Qualifier section below. Following the qualifier descriptions, there is a detailed discussion on linker options files and their qualifiers.

Qualifiers

/DEBUG
/NODEBUG (default)

To use the debugger, you must compile, or assemble and link program units using the /DEBUG qualifier. This ensures that source debugging information is properly stored for later use by the debugger. This qualifier controls the generation of debug symbol information for use with the debugger. The information is in the form of a link map that associates machine addresses with the symbol names found in a compilation unit. The debugger uses the link map to locate the address of the beginning of a compilation unit and the addresses of source lines and link names.

A program that you want to run with the debugger must be linked with the /DEBUG qualifier. If supported by the chosen load module format, /DEBUG may also cause symbol information to be output in the load module. The qualifier is ignored if you select /OFORM. In the standard configuration of the 80960 ACS system, none of the outputs support symbol information in the load module. The default is /NODEBUG.

/ENABLE_TRACEBACK
/NOENABLE_TRACEBACK (default)

In the unlikely event that you should receive an Unexpected Error Condition message, you should contact Customer Support. Customer Support may request that you provide additional information about the error condition by including the /ENABLE_TRACEBACK qualifier in the compiler invocation that fails. This qualifier allows the tool to display the exception traceback associated with the unexpected error condition. The information provided in the traceback will allow Customer Support to diagnose the problem more efficiently.

/EXCLUDED
/NOEXCLUDED (default)

The /EXCLUDED qualifier is used with the linker to insert a list of excluded subprograms into the link map listing. The default is /NOEXCLUDED.

/EXCLUDED is a subqualifier of the /MAP linker qualifier. The /MAP qualifier controls the generation and format of the listing map files that the linker produces. With the /EXCLUDED subqualifier, the /MAP qualifier generates a section of the link map that lists Ada subprograms that have been excluded from the linked object file. These subprograms were excluded because they were not used in the call graph of the main program that is being linked.

/FORMAT=<obj_format>
/FORMAT=COFF960 (default)

This qualifier specifies that an object module file of a specified format is to be linked. The object module format is specified as a parameter to this qualifier. You may specify one of the following object module formats for <obj_mod>:

<obj_format>	COFF960 XCOFF960

COFF960 With the linker, /FORMAT=COFF960 specifies that the format of the linker's load module output should be Intel 80960 COFF. It is the default output format generated by the linker.

XCOFF960 When you are linking programs for 80960 Extended Architecture, /FORMAT=XCOFF960 causes the linker to produce a load module in extended COFF format. The default is /FORMAT=COFF960, the Intel 80960 COFF format.

/IMAGE
/NOIMAGE (default)

The /IMAGE qualifier is used with the /MAP qualifier or MAP option in the linker. /IMAGE generates a memory image listing in addition to the link map listing generated by /MAP. The linker writes the image listing to the same file as the link map listing. This is the only optional section of the listing. The default is /NOIMAGE.

In a memory image listing, each nonsequential section of the image in memory starts on a new page. The image listing contains the locations in memory that are printed as hexadecimal values. Each line in the listing is filled with the amount of data on a line that is a multiple of 16 bytes and up to the specified or default WIDTH limit.

Relocatable control sections are printed with a location that is relative to the start of the control section.

/LIBFILE=<file>
/LIBFILE=LIBLST (default)

By default, the library file named LIBLST with a default type of .ALB is used by the 80960 ACS tool set to determine which set of sublibraries are to be referred to during the operation of the tool. This file must be present in the working directory. With the /LIBFILE qualifier, you can specify a library other than the default, LIBLST. The /TEMPLIB qualifier may also be used to create an alternative library. However, the /TEMPLIB and /LIBFILE qualifiers are mutually exclusive; only one or the other qualifier may be used at the same time.

When you specify the /LIBFILE qualifier, you indicate the file specification of an alternative library file that contains the list of sublibraries and optional comments. If you do not specify a file type with the file name, the system uses the file type .ALB.

For example, consider a library file named WORKLIB.ALB with the contents:

```
Name: MYWORK
Name: [CALCPROJ]CALCLIB
Name: TSADA$E960:[LIB]RTL_960.SUB
```

You could specify

```
$ TSADA/E960 LINK/LIBFILE=WORKLIB
```

As an alternative to using /LIBFILE, you may assign the library file specification to the logical name LIBLST. For example,

```
$ ASSIGN WORKLIB.ALB LIBLST
```

/LOAD_MODULE[=<file>]
/LOAD_MODULE=<main>.CF (default)

This qualifier is used with the linker to specify the VMS file name for the load module output created by the linker.

The <file> is the optional VMS file specification for the output. If <file> does not include an file type, the linker will append the .CF file type. If you do not specify an output file, the linker writes the linked output to:

```
<main>.CF
```

The <main> is the Ada name of the main program unit (if present), the name specified as the command line parameter, or the name specified as the first INPUT option, modified as necessary to form a valid VMS file specification.

You can use the /LOAD_MODULE qualifier with the /OPTIONS qualifier. Any output file specification present in the options file is superceded by the specification on the command line. If the /LOAD_MODULE qualifier is used with the /OFORM qualifier, both formats will be produced.

/LOCALS
/NOLOCALS (default)

This qualifier includes local symbols in the link map symbol listing.

/MAP[=<file>]
/NOMAP (default)

This qualifier is used to request and control a link map listing. The format of the link map listing file is described in the *User Guide*.

When you specify this option, you can optionally list a <file>. This is the optional VMS file specification for the output. If you do not specify a file type, the linker uses a default file type of .MAP. If you do not specify an output file, the linker writes the listing to

```
<unit>.MAP
```

where

<unit> Represents the name of the main program unit (if present), the name specified as the command line parameter, or the name specified as the first INPUT option, modified as necessary to form a valid VMS file

specification.

You control the output of the MAP qualifier using one or more of the following qualifiers:

/[NO]IMAGE
/[NO]LOCALS
/[NO]EXCLUDED
/WIDTH=<132 | 80>
/PAGE=<66 | n>

/IMAGE generates a memory image listing in addition to the map listing. The Linker writes the image listing to the same file as the link map listing. This is the only optional section of the listing. The default is /NOIMAGE.

/LOCALS includes local symbols in the link map symbol listing. The default is /NOLOCALS.

/EXCLUDED inserts a list of excluded subprograms into the link map listing. The default is /NOEXCLUDED.

/WIDTH specifies the width of the lines in the listing file. The default value is 132 characters. The alternate width is 80 characters.

/PAGE specifies the number of lines per listing page. The default is 66. You may specify a positive integer greater than 10.

A command line /MAP qualifier supercedes any MAP options in an options file. The default is /NOMAP. /NOMAP can be used on the command line to suppress MAP options specified in an options file.

/MONITOR
/NOMONITOR (default)

Normally, the only visible output produced by the 80960 ACS tool set during operation is error or warning messages. The /MONITOR qualifier enables the reporting of version numbers and messages that allow you to monitor the tool's progress. When you specify /MONITOR, the output is sent to standard output (SYS$OUTPUT).

/OFORM[=<lib_comp>]
/NOOFORM (default)

This qualifier specifies that one output of the linker is to be linked OF. Linked OF is suitable for incomplete modules and can be used subsequently as input to the Ada linker. The linked OF is put into the library as an object form module (OFM) component.

<lib_comp> Represents a library component name.

<unit> The Ada name of the main program unit (if present), the name specified as the command line parameter, or the name specified as the first INPUT option. The output of the link is put into the library as the object form module called <unit> if you have not specified a library component name.

Note that the object form module of <unit> is a library component separate from that of the specification or body of the unit.

If an object form module library component with the specified name already exists in the current working sublibrary, that component is deleted and replaced by the new output.

The /OFORM qualifier may be used with the /OPTIONS qualifier. Any format or name present in the options file is superceded by the format and name specified on the command line. You may request /OFORM instead of the default /EXECUTE_FORM, or in addition to a load module format. To obtain both an object form module and a load module, you must enter both qualifiers.

/OPTIONS[=<file>]
/NOOPTIONS (default)

/OPTIONS specifies that the linker is to process additional options obtained interactively, or from a linker options file.

<file> This is a valid VMS file specification. It represents a file that contains linker options. If no file type is present, the linker uses the default file type .OPT.

The default file specification is SYS$INPUT. The default is /NOOPTIONS.

The /LOAD_MODULE qualifier and the /OFORM qualifier may be used with the /OPTIONS qualifier. Any format present in the options file is superceded by the format specified on the command line when you use /LOAD_MODULE with /OPTIONS. When /OFORM is used with /OPTIONS, any output file specification present in the options file is superceded by the specification on the command line.

Linker options files are discussed in this chapter, in section 1.4, "The linker options file." For specific linker options file commands, see section 1.4.2, "Linker options and their qualifiers."

/PAGE=<n>
/PAGE=66 (default)

With the LINK command, /PAGE specifies the number of lines per listing page. The default is 66. You may specify a positive integer greater than 10.

/SYMBOL_FILE[=<file>]
/NOSYMBOL_FILE (default)

This qualifier produces a file that contains all of the global symbols used in the link. The default file name is <main>.SYM., with a default file type of .SYM. This file provides you with a simple means of obtaining information about symbol names and values.

The <main>.SYM file is an ascii file that contains one entry per line. An entry consists of the following format:

```
NNNNNNNNTAAAAAAAAAAA
```

Where

NNNNNNNN An 8 character ascii represention of the value of the symbol.

T A one character ascii represention of the type of unit in which the symbol was located. The characters have the following meaning:

```
"=" => Ada_Unit
"*" => Ofm_Unit
"#" => Link time defined symbol
```

AAAAAAAA The ascii representation of the symbol (truncated, if necessary, to a maximum of 200 characters).

/TEMPLIB=(<sublib>[,...])
/LIBFILE=LIBLST (default)

This qualifier allows you to define a temporary library consisting of a selection of sublibraries. The temporary library may be used for the duration of a single command. In all uses, the /TEMPLIB and /LIBFILE options are mutually exclusive; only one or the other qualifier may be used at the same time.

<sublib> The name of the sublibrary, optionally prefixed with the specification of the VMS directory in which it resides. If no directory is specified, the current default directory is assumed. Multiple sublibrary file specifications are separated by commas in a list.

Semantically, the argument string of this qualifier is the logical equivalent of a library file containing the listed sublibraries, one per line, in the order listed. Thus, we could list the sublibraries:

```
/TEMPLIB=(MYWORK,[CALCPROJ]CALCLIB,TSADA$E960:[LIB]RTL_E960.SUB)
```

When used with the linker, this qualifier specifies a list of sublibraries to be used for a single run of the linker. If you do not specify /LIBFILE or /TEMPLIB, the linker assumes that the library is specified by the library file named LIBLST in the current working directory.

/VIRARS_SIZE=<n>
/VIRARS_SIZE=1000 (default)

This qualifier specifies the amount of space, in kilobytes, of buffer space to be allocated for the linker. You must specify a value for the size when you use /VIRARS_SIZE. The default amount of space is 1000 kilobytes.

/VIRTUAL_SPACE=<n>
/VIRTUAL_SPACE=1000 (default)

This qualifier specifies the number of 1kb pages that will be used in memory while the tool executes. Greater values will usually improve performance, but will result in more physical memory requirements.

/WIDTH=<n>
/WIDTH=132 (default)

This qualifier, used with the linker, specifies the width of the lines in the listing file. The default value is 132 characters. The alternate width is 80 characters.

<n> The width of the lines in the listing file. The value can be 132 or 80 characters.

1.4. The linker options file

The /OPTIONS command line qualifier is used to specify that the linker is to read in additional options from an options file or from SYS$INPUT. An options file is a text file that contains linker commands to be interpreted by LINK. You specify the name of the options file on the command line with the /OPTIONS qualifier. The default file specification is SYS$INPUT, so the linker may read commands from a command file or from the terminal immediately after you enter the linker command line. If you are entering the options interactively, there are no prompts on the screen. To end your input, you enter an EXIT command. Options cannot be entered both interactively and in an options file.

There is a basic linker options file provided with this release. It is tailored to your target. You may obtain additional options files for more targets.

1.4.1. Options file format

You must enter each linker option on a separate line. To continue to the next line, use the standard VMS continuation character (-). Comments may be entered with the VMS comment (!) or the Ada comment (--). Options and qualifiers may be abbreviated to as few letters as needed to make a unique abbreviation. The linker reads and processes options until it finds an EXIT or QUIT command, or an end-of-file marker. For more information on the linker options file format, see the "Linker" chapter in *User Guide, Part B*.

1.4.2. Linker options and their qualifiers

The available linker commands and options are summarized in the table below. Defaults are shown in italics.

Table 1-1. Linker options and their qualifiers

DEFINE /<symbol> = <value> [/ADDRESS]	-- Specify link-time values for symbols.
EXIT	-- Terminate options list.
INPUT [/MAIN | /*SPEC* | /BODY | /OFM] [/EXPORT_DEFINITIONS] [/PHANTOM | /WORKING_SUBLIB] [/NOSEARCH] <library_component_name>	-- Identify object modules to be linked and specify the search path.
LOCATE [/CONTROL_SECTION=CODE | DATA | LTRL | STACK] [/COMPONENT_NAME=<library_component_name> [/*SPEC* | /BODY | /OFM | /MAIN | /ASM]] [/AT=<address>] [/IN=<region>] [/AFTER=<control_section> | <library_component_name>] [/ALIGNMENT=<value>]	-- Specify addresses for control sections.
MAP [/[*NO*]IMAGE] [/[*NO*]LOCALS] [/[*NO*]EXCLUDED] [/WIDTH=<*132* | 80>] [/LINES_PER_PAGE=<n>] (*50*) [<file>]	-- Control link map generation.
OUTPUT [/*COMPLETE* | /INCOMPLETE] [/*LOAD_MODULE*[=<file>] [/OBJECT_FORM[=<library_component_name>]]	-- Specify complete or incomplete output and its format.
QUIT	-- Abandon link operation.
REGION /LOW_BOUND=<address> /HIGH_BOUND=<address> [/UNUSED] [<region>]	-- Define and name memory regions.

1.4.3. DEFINE: Defining global symbols

This command allows you to define global symbols referenced in the Ada CGS and other imported OF modules. Multiple DEFINE commands can be used. The format is:

```
DEFINE/<symbol_name>=<value>[/ADDRESS]
```

where:

`<symbol_name>`	A name that you specify, or one of the standard CGS names.
`<value>`	Specified by the user. <value> can be specified by using a decimal value, a hexadecimal-based literal in Ada syntax (16#hex#), or an unsigned hexadecimal value in the format (%Xhex).
`/ADDRESS`	Specifies that the value input is to be used as an address rather than a signed long integer value. This option is necessary only when address values are 16#80000000# or larger.

1.4.4. EXIT: Ending user input

This option may be used to specify the end of user input within the options file: no options are processed after the EXIT option. An end-of-file is equivalent to the EXIT option. The format is:

```
EXIT
```

1.4.5. INPUT: Specifying the input

This command specifies the name of the Ada library unit to be linked, its usage, and its search path. Multiple INPUT options may be used.

By default, the object associated with the specified name is included in the linked output. If the link is incomplete, the symbols defined in the OF modules will not be exported as global definitions. The /EXPORT_DEFINITIONS option may be used to override this default. In complete linkage, all references are to be resolved, so the linker does not export global definitions and ignores this option. The /MAIN and /PHANTOM options may be used to specify special usage for the input OF modules.

The use of an INPUT option does not prevent an *unreferenced* Ada unit from being excluded from the module because of unused subprogram elimination. If an unreferenced unit needs to be included in the module, such as to provide an interrupt handler, it can be done by linking in two steps.

First, include the unreferenced unit in an incomplete link. Then do a complete link that inputs the result of the incomplete link.

When the default search path for a specified library unit is used, the linked output includes all compilation units comprising the extended family of the unit that exist in the current library. The search path options /NOSEARCH and /WORKING_SUBLIB may be used to override the default and include only the unit itself or only those entities of the extended family in the working sublibrary.

The format of the INPUT command is:

```
INPUT[/MAIN] [/SPEC| | /BODY | /OFM]
   [/EXPORT_DEFINITIONS]
   [/PHANTOM | /WORKING_SUBLIB]
   [/NOSEARCH] <comp_unit>
```

where:

<comp_unit> The name of the library compilation unit. The following options specify the kind of library compilation unit input:

qualifier	library component
/SPEC	library unit
/BODY	secondary unit
/OFM	linked OF
/ASM	assembled program unit
/MAIN	main program library unit

/SPEC should be used only for library units without bodies, since those are the only library units that have object associated with them. All other compilation units should be included in the link using either /BODY or /MAIN, if the unit is a main program unit. In a complete link, using /SPEC or /BODY makes sense only when combined with /WORKING_SUBLIB since the library or secondary unit will automatically be brought into the link if needed and excluded if not needed.

/MAIN Specifies that the library unit is to be the main program in the link. The library unit name specified with the first INPUT command is used as the default OUTPUT and MAP name, unless you specify other names. Specification of both /OFM and /MAIN is an error.

/EXPORT_DEFINITIONS	Used with incomplete linking. It indicates that the global symbols defined in the input unit are exported as global definitions. Thus, the interface remains visible to the user of the incomplete linked output. The definitions of any imported units that are included by the library search are not exported as global definitions, however. The linker ignores this option in complete linkages.
/PHANTOM	Specifies that the symbols defined in the object associated with the input may be used to resolve references in the linked output. The linker does not generate executable object for the phantom module or include it in the linked output. The library unit specified with a phantom input unit must be an incomplete linked object that has no unresolved references. Using the /WORKING_SUBLIB option with a phantom input file is an error. The default search action for a phantom input file is /NOSEARCH.
/WORKING_SUBLIB	Specifies that only those extended family entities in the current working sublibrary are to be included in the linked output.
/NOSEARCH	Specifies that only the specified input unit is to be included in the linked output. For complete links, this option is only effective for linked or imported OF, as when used in combination with /OFM.

When the name of the OUTPUT and MAP files are not specified, the linker names the files after the first INPUT command's named unit.

1.4.6. LOCATE: Specifying locations of control sections

This command specifies the location of a control section in physical memory or within a named region of physical memory. The format is:

```
LOCATE
   [/CONTROL_SECTION=CODE | DATA | UDATA | MAP]
   [/COMPONENT_NAME=<comp_unit>
      [/SPEC | /BODY | /OFM]]
   [/AT=<address>]
   [/IN=<region_name>]
   [/AFTER=<csect_name | comp_unit>]
   [/ALIGNMENT=<value>]
```

where:

/CONTROL_SECTION Specifies the name of the control section to be located. The valid control section names for the 80960 are CODE, DATA, LTTR and STACK. If a control section name is omitted and a compilation unit name is given, none of the control sections are explicitly located.

/COMPONENT_NAME Specifies the name of the library compilation unit that is to be located and has one of the following options:

```
/SPEC -- library unit,
/BODY -- secondary unit,
/OFM  -- linked OF or imported unit,
/ASM  -- assembled unit, or
/MAIN -- main program unit.
```

The only Ada compilation units that have object associated with the library unit are body-less compilation units. All others should be located using the /BODY option. If no /COMPONENT_NAME option is used with the LOCATE command, the linker assumes that LOCATE refers to all compilation units not explicitly located by other LOCATE commands.

/AT Specifies that the image is to be located at the specified memory location. The address specified must be a valid 80960 memory location. The <address> can be specified by using a decimal value, a hexadecimal-based literal in Ada syntax (16#hex#), or an unsigned hexadecimal value in VMS format (%Xhex).

/IN Specifies that the image is to be located in the first available location in the named memory region. The memory region, <region_name> must have been defined by a previous REGION command. This option may be used in conjunction with the /AFTER option to locate compilation units in a specific order in a memory region and/or with the /ALIGNMENT option to specify alignment.

/AFTER Specifies that the image is to be located immediately after the specified control section or compilation unit (using default alignment or /ALIGNMENT). The specified control section or compilation unit must have been located explicitly in a previous LOCATE command.

/ALIGNMENT Specifies the byte alignment of the image, if any. If no alignment is specified, control sections on the MCMIL are word-aligned.

The options /IN, /AFTER, and /ALIGNMENT can be used in any combination. The use of /IN, /AFTER, or /ALIGNMENT in the same LOCATE command as the /AT option is an error.

A LOCATE command with no compilation unit name and no control section name is equivalent to a /BASE command line qualifier, i.e., all non-located sections are located starting at the base specified.

Control sections not designated in a LOCATE command are placed at the first available location after the default location, if any. An incomplete link with no default location will cause the control sections in the output to remain relocatable.

1.4.7. MAP: Producing a link map listing

This option is used to request and control a **link map listing**. The format of the link map listing file is described in the *User Guide*. This option operates in the same way as the /MAP command line qualifier. The format is:

```
MAP
    [/[NO]IMAGE]
    [/[NO]INCLUDE_LOCALS]
    [/[NO]EXCLUDED]
    [/WIDTH=<132 | 80>]
    [/HEIGHT=<value>]
    [<file>]
```

where:

/IMAGE	Generates a memory image listing.
/INCLUDE_LOCALS	Includes local symbols in the link map symbol listing. The default is /NOLOCALS.
/EXCLUDED	Inserts a list of excluded subprograms into the link map listing. The default is /NOEXCLUDED.
/WIDTH	Specifies the width of the lines in the listing file. The default value is 132 characters. The alternate width is 80 characters.
/HEIGHT	Specifies the number of lines per listing page. The default is 50. You may specify a positive integer greater than 10.
<file>	Allows you to to override the default of sending the output to <unit>.MAP

A command line /MAP qualifier supercedes any MAP qualifiers in an options file.

1.4.8. OUTPUT: Specifying the output

This command specifies the type, format, and name of the linked output. Only one OUTPUT option may be used. The format is:

```
OUTPUT[/COMPLETE | /INCOMPLETE]
   [/LOAD_MODULE=[<file>]]
   [/OBJECT_FORM=[<comp_unit>]]
```

where:

/COMPLETE	Specifies that complete modules are to be generated. /COMPLETE is the default. When the output type is complete, the linker reports any unresolved references as errors.
/INCOMPLETE	Specifies that an incomplete module is to be generated, either as a load module or as OF. A linked OF module is incomplete if it (1) contains unresolved references that will be resolved in subsequent /LINK operations or (2) is a fully resolved module that does not contain an entry point. A module that is incomplete because it does not contain an entry point may be output as a load module or as OF. Incomplete modules with unresolved references must be output in OF.

/LOAD_MODULE Specifies that one output of the linker should be an executable load module. <file> is the optional VMS file specification for the output. If <file> does not include a file type, the linker appends a .CF file type to the file specification. If you do not specify an output file, the linker writes the output of the link to

```
<unit>.CF
```

where <unit> is the Ada name of the main program unit (if present), the name specified as the command line parameter, or the name specified as the first INPUT option, modified as necessary to form a valid VMS file specification.

/OBJECT_FORM Specifies that the output of the linker is to be linked OF. Linked OF is output in the Ada library as an object form module unit. <comp_unit> is any valid library unit name. If you do not specify an output file, the linker puts the output of the link into the library as the object form module, <unit> where <unit> is the Ada name of the main program (if present), the name specified as the command line parameter, or the name specified as the first INPUT option. Incomplete linked OF files may be used as input in subsequent linker operations.

The linker can generate a load module, an OF module, or both. For example, incomplete objects that will be used as phantoms must be output both as a load module for downloading and as OFM for subsequent linkage.

Specification of both /COMPLETE and /INCOMPLETE is an error.

1.4.9. QUIT: Terminating options file input and linking

This option is used to terminate interactive options input and abort the /LINK operation. The format is:

```
QUIT
```

1.4.10. REGION: Specifying a range of memory

This command names a memory region and specifies its location in physical memory. This named memory region may be used in subsequent LOCATE commands. In addition, the REGION command may be used to specify an unused memory region.

Multiple REGION options may be used. The format of the command is:

```
REGION
   /LOW_BOUND=<address>
   /HIGH_BOUND=<address>
   [/UNUSED]
   [<region_name>]
```

where:

<region_name>	A user-specified name for a region of memory.
/LOW_BOUND /HIGH_BOUND	The bounds of the memory region. These options and associated values are mandatory. An <address> can be specified as a decimal value, a hexadecimal-based literal in Ada syntax (16#hex#), or an unsigned hexadecimal value in VMS format (%Xhex).
/UNUSED	Specifies that the linker is not to use this memory region. Unused memory regions do not need to be named.

1.4.11. Example linker options file: LINK.OPT

This linker options file is used to verify the installation of the linker. This options file is also located in TSADA$E960:[TARGET.MCMIL.EXAMPLES].

```
TARGET 180960MC
--
INPUT/OFM/PHANTOM MCMIL_PHANTOM
--
CONTROL_SECTION/DATA BSS
--
LOCATE/CONTROL_SECTION=CODE/AT=16#180d0000#
LOCATE/CONTROL_SECTION=DATA/AFTER=CODE
--
-- MAP
```

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are given on the following customer supplied page.

```
INTEGER:

        size            = 32
        'first          = -2147483648
        'last           =  214748367


SHORT_INTEGER:

        size            = 16
        'first          = -32768
        'last           =  32767


SHORT_SHORT_INTEGER:

        size            = 8
        'first          = -128
        'last           =  127


FLOAT:

        size            = 32
        digits          = 6
        'first          = -3.40282E+38
        'last           = +3.40282E+38
        machine_radix   = 2
        machine_mantissa = 24
        machine_emin    = -125
        machine_emax    = +128


LONG_FLOAT:

        size            = 64
        digits          = 15
        'first          = -1.7976931349E+308
        'last           = +1.7976931349E+308
        machine_radix   = 2
        machine_mantissa = 53
        machine_emin    = -1024
        machine_emax    = +1024


LONG_LONG_FLOAT:

        size            = 96
        digits          = 18
        'first          = -1.18973149535723177E+4932
        'last           = +1.18973149535723177E+4932
        machine_radix   = 2
        machine_mantissa = 64
        machine_emin    = -16381
        machine_emax    = +16384


DURATION:

        size            = 32
        delta           = 6.10351562500000E-005
        first           = -86400
        last            = +86400
```

TELESOFT

i960 ACS Ada Development System for VAX/VMS to Embedded i960 Targets

Attachment H

Appendix F of the Ada Language Reference Manual

APF-1940N-V1.1 02DEC91

Version 4.1

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the rights in Technical Data and Computer Software clause at DFAR 252.227-7013, or FAR 52.227-14, ALT III and/or FAR 52.227-19 as set forth in the applicable Government Contract.

TeleSoft
5959 Cornerstone Court West
San Diego, CA 92121-9819
(619) 457-2700
(Contractor)

Table of Contents

Appendix F of the Ada LRM

1. Introduction

The i960 ACS compiler compiles the full ANSI Ada language as defined by the *Reference Manual for the Ada Programming Language* (LRM) (ANSI/MIL-STD-1815A). This appendix describes the characteristics of the compiler and run-time environment that are designated by the LRM as implementation dependent. These language-related issues are presented in the order in which they appear in the LRM Appendix F.

2. Implementation-dependent pragmas

i960 ACS has the following implementation-dependent pragmas:

pragma Comment
pragma Export
pragma Images
pragma Interface_Information
pragma No_Suppress
pragma Preserve_Layout
pragma Suppress_All

2.1. Pragma Comment

Pragma Comment is used for embedding a comment into the object code. The syntax is

```
pragma Comment ( <string_literal> );
```

where <string_literal> represents the characters to be embedded in the object code. Pragma Comment is allowed only within a declarative part or immediately within a package specification. Any number of comments may be entered into the object code by use of pragma Comment.

2.2. Pragma Export

Pragma Export enables you to export an Ada subprogram or object to either the C language or assembly. The pragma is not supported for Pascal or FORTRAN. The syntax is

```
pragma Export ( [ Name => ] <subprogram_or_object_name>
                [, [ Link_Name => ] <string_literal> ]
                [, [ Language  => ] <identifier> ] );
```

2.3. Pragma Images

Pragma Images controls the creation and allocation of the image and index tables for a specified enumeration type. The syntax is

```
pragma Images(<enumeration_type>, Deferred);

pragma Images(<enumeration_type>, Immediate);
```

2.4. Pragma Interface_Information

Pragma Interface_Information provides information for the optimizing code generator when interfacing non-Ada languages or doing machine code insertions. Pragma Interface_Information is always associated with a pragma Interface except for machine code insertion procedures, which do not use a preceding pragma Interface. The syntax of the pragma is

```
pragma Interface_Information
  (Name,             -- Ada subprogram, required
   Link_Name,        -- string, default = ""
   Mechanism,        -- string, default = ""
   Parameters,       -- string, default = ""
   Regs_Clobbered); -- string, default = ""
```

2.5. Pragma No_Suppress

Pragma No_Suppress is a i960 ACS-defined pragma that prevents the suppression of checks within a particular scope. It can be used to override pragma Suppress in an enclosing scope. The pragma uses the same syntax and can occur in the same places in the source as pragma Suppress. The syntax is

```
pragma No_Suppress (<identifier> [, [ON =>] <name>]);
```

<identifier> The type of check you do not want to suppress.

<name> The name of the object, type/subtype, task unit, generic unit, or subprogram within which the check is to be suppressed. <name> is optional.

2.6. Pragma Preserve_Layout

The i960 ACS compiler reorders record components to minimize gaps within records. Pragma Preserve_Layout forces the compiler to maintain the Ada source order of components of a given record type, thereby preventing the compiler from performing this record layout optimization.

The syntax of this pragma is

```
Pragma Preserve_Layout ( ON => <record_type> );
```

Preserve_Layout must appear before any forcing occurrences of the record type and must be in the same declarative part, package specification, or task specification. This pragma can be applied to a record type that has been packed. If Preserve_Layout is applied to a record type that has a record representation clause, the pragma only applies to the components that do not have component clauses. These components will appear in Ada source order after the components with component clauses.

2.7. Pragma Suppress_All

Suppress_All is an implementation-defined pragma that suppresses all checks in a given scope. Pragma Suppress_All takes no arguments and can be placed in the same scopes as pragma Suppress.

In the presence of pragma Suppress_All or any other Suppress pragma, the scope that contains the pragma will have checking turned off. This pragma should be used in a safe piece of time-critical code to allow for better performance.

3. Implementation-dependent attributes

i960 ACS has the following implementation-dependent attributes:

'Offset *(in MCI)*
'Subprogram_Value
'Extended_Image
'Extended_Value
'Extended_Width
'Extended_Aft
'Extended_Digits
'Extended_Fore

3.1. 'Offset

'Offset yields the offset of an Ada object from its parent frame. This attribute supports machine code insertions.

3.2. 'Subprogram_Value

This attribute returns the value of the record type Subprogram_Value defined in package System.

3.3. Extended attributes for scalar types

The extended attributes extend the concept behind the text attributes 'Image, 'Value, and 'Width to give the user more power and flexibility when displaying values of scalars. Extended attributes differ in two respects from their predefined counterparts:

1. Extended attributes take more parameters and allow control of the format of the output string.

2. Extended attributes are defined for all scalar types, including fixed and floating point types.

Named parameter associations are not currently supported for the extended attributes.

Extended versions of predefined attributes are provided for integer, enumeration, floating point, and fixed point types:

Integer	Enumeration	Floating Point	Fixed Point
'Extended_Image	'Extended_Image	'Extended_Image	'Extended_Image
'Extended_Value	'Extended_Value	'Extended_Value	'Extended_Value
'Extended_Width	'Extended_Width	'Extended_Digits	'Extended_Fore
			'Extended_Aft

For integer and enumeration types, the 'Extended_Value attribute is identical to the 'Value attribute. For enumeration types, the 'Extended_Width attribute is identical to the 'Width attribute.

The extended attributes can be used without the overhead of including Text_IO in the linked program. The following examples illustrate the difference between instantiating Text_IO.Float_IO to convert a float value to a string and using Float'Extended_Image:

```
with Text_IO;
function Convert_To_String ( Fl : Float ) return String is
  Temp_Str : String ( 1 .. 6 + Float'Digits );
package Flt_IO is new Text_IO.Float_IO (Float);
begin
  Flt_IO.Put ( Temp_Str, Fl );
  return Temp_Str;
end Convert_To_String;


function Convert_To_String_No_Text_IO( Fl : Float ) return String i
begin
  return Float'Extended_Image ( Fl );
end Convert_To_String_No_Text_IO;


with Text_IO, Convert_To_String, Convert_To_String_No_Text_IO;
procedure Show_Different_Conversions is
  Value : Float := 10.03376;
begin
  Text_IO.Put_Line ( "Using the Convert_To_String, the value of
  the variable is : " & Convert_To_String ( Value ) );
  Text_IO.Put_Line ( "Using the Convert_To_String_No_Text_IO,
  the value is : " & Convert_To_String_No_Text_IO ( Value ) );
end Show_Different_Conversions;
```

3.3.1. Integer attributes

'Extended_Image

```
X'Extended_Image(Item,Width,Base,Based,Space_If_Positive)
```

Returns the image associated with Item as defined in Text_IO.Integer_IO. The Text_IO definition states that the value of Item is an integer literal with no underlines, no exponent, no leading zeros (but a single zero for the zero value), and a minus sign if negative. If the resulting sequence of characters to be output has fewer than Width characters, leading spaces are first output to make up the difference. (LRM 14.3.7:10,14.3.7:11)

For a prefix X that is a discrete type or subtype, this attribute is a function that may have more than one parameter. The parameter Item must be an integer value. The resulting string is without underlines, leading zeros, or trailing spaces.

Parameters

`Item`	The item for which you want the image; it is passed to the function. Required.
`Width`	The minimum number of characters to be in the string that is returned. If no width is specified, the default (0) is assumed. Optional.
`Base`	The base in which the image is to be displayed. If no base is specified, the default (10) is assumed. Optional.
`Based`	An indication of whether you want the string returned to be in base notation or not. If no preference is specified, the default (false) is assumed. Optional.
`Space_If_Positive`	An indication of whether or not a positive integer should be prefixed with a space in the string returned. If no preference is specified, the default (false) is assumed. Optional.

Examples

```
subtype X is Integer Range -10..16;
```

Values yielded for selected parameters:

```
X'Extended_Image(5)          = "5"
X'Extended_Image(5,0)        = "5"
X'Extended_Image(5,2)        = " 5"
X'Extended_Image(5,0,2)      = "101"
```

```
X'Extended_Image(5,4,2)                 = " 101"
X'Extended_Image(5,0,2,True)            = "2#101#"
X'Extended_Image(5,0,10,False)          = "5"
X'Extended_Image(5,0,10,False,True)     = " 5"
X'Extended_Image(-1,0,10,False,False)   = "-1"
X'Extended_Image(-1,0,10,False,True)    = "-1"
X'Extended_Image(-1,1,10,False,True)    = "-1"
X'Extended_Image(-1,0,2,True,True)      = "-2#1#"
X'Extended_Image(-1,10,2,True,True)     = "      -2#1#"
```

'Extended_Value

```
X'Extended_Value(Item)
```

Returns the value associated with Item as defined in Text_IO.Integer_IO. The Text_IO definition states that given a string, it reads an integer value from the beginning of the string. The value returned corresponds to the sequence input. (LRM 14.3.7:14)

For a prefix X that is a discrete type or subtype, this attribute is a function with a single parameter. The actual parameter Item must be of predefined type string. Any leading or trailing spaces in the string X are ignored. In the case where an illegal string is passed, a Constraint_Error is raised.

Parameter

`Item` A parameter of the predefined type string; it is passed to the function. The type of the returned value is the base type X. Required.

Examples

```
subtype X is Integer Range -10..16;
```

Values yielded for selected parameters:

```
X'Extended_Value("5")        = 5
X'Extended_Value(" 5")       = 5
X'Extended_Value("2#101#")   = 5
X'Extended_Value("-1")       = -1
X'Extended_Value(" -1")      = -1
```

'Extended_Width

```
X'Extended_Width(Base,Based,Space_If_Positive)
```

Returns the width for subtype of X. For a prefix X that is a discrete subtype, this attribute is a function that may have multiple parameters. This attribute yields the maximum image length over all values of the type or subtype *X*.

Parameters

`Base` The base for which the width will be calculated. If no base is specified, the default (10) is assumed. Optional.

`Based` An indication of whether the subtype is stated in based notation. If no value for based is specified, the default (false) is assumed. Optional.

`Space_If_Positive` An indication of whether or not the sign bit of a positive integer is included in the string returned. If no preference is specified, the default (false) is assumed. Optional.

Examples

```
subtype X is Integer Range -10..16;
```

Values yielded for selected parameters:

```
X'Extended_Width                  = 3  -- "-10"
X'Extended_Width(10)              = 3  -- "-10"
X'Extended_Width(2)               = 5  -- "10000"
X'Extended_Width(10,True)         = 7  -- "-10#10#"
X'Extended_Width(2,True)          = 8  -- "2#10000#"
X'Extended_Width(10,False,True)   = 3  -- " 16"
X'Extended_Width(10,True,False)   = 7  -- "-10#10#"
X'Extended_Width(10,True,True)    = 7  -- " 10#16#"
X'Extended_Width(2,True,True)     = 9  -- " 2#10000#"
X'Extended_Width(2,False,True)    = 6  -- " 10000"
```

3.3.2. Enumeration type attributes

'Extended_Image

```
X'Extended_Image(Item,Width,Uppercase)
```

Returns the image associated with Item as defined in Text_IO.Enumeration_IO. The Text_IO definition states that given an enumeration literal, it will output the value of the enumeration literal (either an identifier or a character literal). The character case parameter is ignored for character literals. (LRM 14.3.9:9)

For a prefix X that is a discrete type or subtype; this attribute is a function that may have more that one parameter. The parameter Item must be an enumeration value. The image of an enumeration value is the corresponding identifier, which may have character case and return string width specified.

Parameters

`Item` The item for which you want the image; it is passed to the function. Required.

`Width` The minimum number of characters to be in the string that is returned. If no width is specified, the default (0) is assumed. If the Width specified is larger than the image of Item, the return string is padded with trailing spaces. If the Width specified is smaller than the image of Item, the default is assumed and the image of the enumeration value is output completely. Optional.

`Uppercase` An indication of whether the returned string is in upper case characters. In the case of an enumeration type where the enumeration literals are character literals, Uppercase is ignored and the case specified by the type definition is taken. If no preference is specified, the default (true) is assumed. Optional.

Examples

```
type X is (red, green, blue, purple);
type Y is ('a', 'B', 'c', 'D');
```

Values yielded for selected parameters:

```
X'Extended_Image(red)              = "RED"
X'Extended_Image(red, 4)           = "RED "
X'Extended_Image(red,2)            = "RED"
X'Extended_Image(red,0,false)      = "red"
X'Extended_Image(red,10,false)     = "red       "
Y'Extended_Image('a')              = "'a'"
Y'Extended_Image('B')              = "'B'"
Y'Extended_Image('a',6)            = "'a'  "
Y'Extended_Image('a',0,true)       = "'a'"
```

'Extended_Value

```
X'Extended_Value(Item)
```

Returns the image associated with Item as defined in Text_IO.Enumeration_IO. The Text_IO definition states that it reads an enumeration value from the beginning of the given string and returns the value of the enumeration literal that corresponds to the sequence input. (LRM 14.3.9:11)

For a prefix X that is a discrete type or subtype, this attribute is a function with a single parameter. The actual parameter Item must be of predefined type string. Any leading or trailing spaces in the string X are ignored. In the case where an illegal string is passed, a Constraint_Error is raised.

Parameter

`Item` A parameter of the predefined type string; it is passed to the function. The type of the returned value is the base type of X. Required.

Examples

```
type X is (red, green, blue, purple);
```

Values yielded for selected parameters:

```
X'Extended_Value("red")          = red
X'Extended_Value(" green")       = green
X'Extended_Value("    Purple")   = purple
X'Extended_Value(" GreEn  ")     = green
```

'Extended_Width

```
X'Extended_Width
```

Returns the width for subtype of X.

For a prefix X that is a discrete type or subtype; this attribute is a function. This attribute yields the maximum image length over all values of the enumeration type or subtype X.

Parameters

There are no parameters to this function. This function returns the width of the largest (width) enumeration literal in the enumeration type specified by X.

Examples

```
type X is (red, green, blue, purple);
type Z is (X1, X12, X123, X1234);
```

Values yielded:

```
X'Extended_Width     = 6  -"purple"
Z'Extended_Width     = 5  -"X1234"
```

3.3.3. Floating point attributes

'Extended_Image

```
X'Extended_Image(Item,Fore,Aft,Exp,Base,Based)
```

Returns the image associated with Item as defined in Text_IO.Float_IO. The Text_IO definition states that it outputs the value of the parameter Item as a decimal literal with the format defined by the other parameters. If the value is negative, a minus sign is included in the integer part of the value of Item. If Exp is 0, the integer part of the output has as many digits as are needed to represent the integer part of the value of Item or is zero if the value of Item has no integer part. (LRM 14.3.8:13, 14.3.8:15)

Item must be a Real value. The resulting string is without underlines or trailing spaces.

Parameters

`Item` The item for which you want the image; it is passed to the function. Required.

`Fore` The minimum number of characters for the integer part of the decimal representation in the return string. This includes a minus sign if the value is negative and the base with the '#' if based notation is specified. If the integer part to be output has fewer characters than specified by Fore, leading spaces are output first to make up the difference. If no Fore is specified, the default value (2) is assumed. Optional.

`Aft` The minimum number of decimal digits after the decimal point to accommodate the precision desired. If the delta of the type or subtype is greater than 0.1, then Aft is 1. If no Aft is specified, the default (X'Digits-1) is assumed. If based notation is specified, the trailing '#' is included in Aft. Optional.

`Exp` The minimum number of digits in the exponent. The exponent consists of a sign and the exponent, possibly with leading zeros. If no Exp is specified, the default (3) is assumed. If Exp is 0, no exponent is used. Optional.

`Base` The base that the image is to be displayed in. If no base is specified, the default (10) is assumed. Optional.

`Based` An indication of whether you want the string returned to be in based notation or not. If no preference is specified, the default (false) is assumed. Optional.

Examples

```
type X is digits 5 range -10.0 .. 16.0;
```

Values yielded for selected parameters:

```
X'Extended_Image(5.0)                    = " 5.0000E+00"
X'Extended_Image(5.0,1)                  = "5.0000E+00"
X'Extended_Image(-5.0,1)                 = "-5.0000E+00"
X'Extended_Image(5.0,2,0)                = " 5.0E+00"
X'Extended_Image(5.0,2,0,0)              = " 5.0"
X'Extended_Image(5.0,2,0,0,2)            = "101.0"
X'Extended_Image(5.0,2,0,0,2,True)       = "2#101.0#"
X'Extended_Image(5.0,2,2,3,2,True)       = "2#1.1#E+02"
```

'Extended_Value

```
X'Extended_Value(Item)
```

Returns the value associated with Item as defined in Text_IO.Float_IO. The Text_IO definition states that it skips any leading zeros, then reads a plus or minus sign if present then reads the string according to the syntax of a real literal. The return value is that which corresponds to the sequence input. (LRM 14.3.8:9, 14.3.8:10)

For a prefix X that is a discrete type or subtype; this attribute is a function with a single parameter. The actual parameter Item must be of predefined type string. Any leading or trailing spaces in the string X are ignored. In the case where an illegal string is passed, a Constraint_Error is raised.

Parameter

`Item`	A parameter of the predefined type string; it is passed to the function. The type of the returned value is the base type of the input string. Required.

Examples

```
type X is digits 5 range -10.0 .. 16.0;
```

Values yielded for selected parameters:

```
X'Extended_Value("5.0")          = 5.0
X'Extended_Value("0.5E1")        = 5.0
X'Extended_Value("2#1.01#E2")    = 5.0
```

'Extended_Digits

```
X'Extended_Digits(Base)
```

Returns the number of digits using base in the mantissa of model numbers of the subtype X.

Parameter

`Base`	The base that the subtype is defined in. If no base is specified, the default (10) is assumed. Optional.

Examples

```
type X is digits 5 range -10.0 .. 16.0;
```

Values yielded:

```
X'Extended_Digits    = 5
```

3.3.4. Fixed point attributes

'Extended_Image

```
X'Extended_Image(Item,Fore,Aft,Exp,Base,Based)
```

Returns the image associated with Item as defined in Text_IO.Fixed_IO. The Text_IO definition states that it outputs the value of the parameter Item as a decimal literal with the format defined by the other parameters. If the value is negative, a minus sign is included in the integer part of the value of Item. If Exp is 0, the integer part of the output has as many digits as are needed to represent the integer part of the value of Item or is zero if the value of Item has no integer part. (LRM 14.3.8:13, 14.3.8:15)

For a prefix X that is a discrete type or subtype; this attribute is a function that may have more than one parameter. The parameter Item must be a Real value. The resulting string is without underlines or trailing spaces.

Parameters

`Item` The item for which you want the image; it is passed to the function. Required.

`Fore` The minimum number of characters for the integer part of the decimal representation in the return string. This includes a minus sign if the value is negative and the base with the '#' if based notation is specified. If the integer part to be output has fewer characters than specified by Fore, leading spaces are output first to make up the difference. If no Fore is specified, the default value (2) is assumed. Optional.

`Aft` The minimum number of decimal digits after the decimal point to accommodate the precision desired. If the delta of the type or subtype is greater than 0.1, then Aft is 1. If no Aft is specified, the default (X'Digits-1) is assumed. If based notation is specified, the trailing '#' is included in Aft. Optional.

`Exp` The minimum number of digits in the exponent; the exponent consists of a sign and the exponent, possibly with leading zeros. If no Exp is specified, the default (3) is assumed. If Exp is 0, no exponent is used. Optional.

`Base` The base in which the image is to be displayed. If no base is specified, the default (10) is assumed. Optional.

`Based` An indication of whether you want the string returned to be in based notation or not. If no preference is specified, the default (false) is assumed. Optional.

Examples

```
type X is delta 0.1 range -10.0 .. 17.0;
```

Values yielded for selected parameters:

```
X'Extended_Image(5.0)                    = " 5.00E+00"
X'Extended_Image(5.0,1)                  = "5.00E+00"
X'Extended_Image(-5.0,1)                 = "-5.00E+00"
X'Extended_Image(5.0,2,0)                = " 5.0E+00"
X'Extended_Image(5.0,2,0,0)              = " 5.0"
X'Extended_Image(5.0,2,0,0,2)            = "101.0"
X'Extended_Image(5.0,2,0,0,2,True)       = "2#101.0#"
X'Extended_Image(5.0,2,2,3,2,True)       = "2#1.1#E+02"
```

'Extended_Value

```
X'Extended_Value(Image)
```

Returns the value associated with Item as defined in Text_IO.Fixed_IO. The Text_IO definition states that it skips any leading zeros, reads a plus or minus sign if present, then reads the string according to the syntax of a real literal. The return value is that which corresponds to the sequence input. (LRM 14.3.8:9, 14.3.8:10)

For a prefix X that is a discrete type or subtype; this attribute is a function with a single parameter. The actual parameter Item must be of predefined type string. Any leading or trailing spaces in the string X are ignored. In the case where an illegal string is passed, a Constraint_Error is raised.

Parameter

`Image`	Parameter of the predefined type string. The type of the returned value is the base type of the input string. Required.

Examples

```
type X is delta 0.1 range -10.0 .. 17.0;
```

Values yielded for selected parameters:

```
X'Extended_Value("5.0")          = 5.0
X'Extended_Value("0.5E1")        = 5.0
X'Extended_Value("2#1.01#E2")    = 5.0
```

'Extended_Fore

```
X'Extended_Fore(Base,Based)
```

Returns the minimum number of characters required for the integer part of the based representation of X.

Parameters

`Base`	The base in which the subtype is to be displayed. If no base is specified, the default (10) is assumed. Optional.
`Based`	An indication of whether you want the string returned to be in based notation or not. If no preference is specified, the default (false) is assumed. Optional.

Examples

```
type X is delta 0.1 range -10.0 .. 17.1;
```

Values yielded for selected parameters:

```
X'Extended_Fore     = 3  -- "-10"
X'Extended_Fore(2)  = 6  -- " 10001"
```

'Extended_Aft

`X'Extended_Aft(Base,Based)`

Returns the minimum number of characters required for the fractional part of the based representation of X.

Parameters

`Base` The base in which the subtype is to be displayed. If no base is specified, the default (10) is assumed. Optional.

`Based` An indication of whether you want the string returned to be in based notation or not. If no preference is specified, the default (false) is assumed. Optional.

Examples

```
type X is delta 0.1 range -10.0 .. 17.1;
```

Values yielded for selected parameters:

```
X'Extended_Aft      = 1  -- "1" from 0.1
X'Extended_Aft(2)   = 4  -- "0001" from 2#0.0001#
```

4. Package System

```
with Unchecked_Conversion;
with Ordinals;

package System is

  --=====================
  -- CUSTOMIZABLE VALUES
  --=====================

  type Name     is (TeleGen2);

  System_Name   : constant name := TeleGen2;

  Memory_Size   : constant := (2 ** 31) -1; --Available memory, in storage u
  Tick          : constant := 1.0 / 100.0;  --Basic clock rate, in seconds

  type Task_Data is    --
    record             -- Adaptation-specific customization information
      null;            -- for task objects.
    end record;        --



  --=============================================================
  -- NON-CUSTOMIZABLE, IMPLEMENTATION-DEPENDENT VALUES
  --=============================================================


  Storage_Unit : constant := 8;
  Min_Int      : constant := -(2 ** 31);
  Max_Int      : constant := (2 ** 31) - 1;
  Max_Digits   : constant := 18;
  Max_Mantissa : constant := 31;
  Fine_Delta   : constant := 1.0 / (2 ** Max_Mantissa);


  subtype Priority is Integer Range 0 .. 31;



  --=====================
  -- ADDRESS TYPE SUPPORT
  --=====================

--
-- The following type, Mixed_Word, is a Phase 1 contract requirement.
-- Ref: SRR dated December 7, 1990 (page 2-4).
-- This type must be be implemented as an access descriptor (AD)
--
  --### type Mixed_Word is private;
```

```
  type Mixed_Word is new Ordinals.Ordinal;

  type Memory is private;

  type Address is access Memory;
--   pragma Access_Address (Address, Virtual);

  type Linear_Address is access Memory;
--   pragma Access_Address (Linear_Address, Linear);

  type AD_Address is access Memory;
--   pragma Access_Address (AD_Address, AD);
    --
    -- Ensures compatibility between addresses and access types.
    -- Also provides implicit NULL initial value.


  function Convert is new Unchecked_Conversion (Ordinals.Ordinal, Address);
  function Convert is new Unchecked_Conversion (Integer, Address);

  Null_Address: constant Address := Convert (Integer'(0));
    --
    -- Initial value for any Address object


  type Address_Value is range -(2**31)..(2**31)-1;
    --
    -- A numeric representation of logical addresses for use in address clauses

--
-- For the 80960MC, we choose not to "define numeric offsets to aid in
-- Address calculations", e.g. Hex_80000000, Hex_90000000, ..., Hex_F0000000
-- These were defined on the 68K; they are judged not vital here.
--

  function Location is new Unchecked_Conversion (Address_Value, Address);
    --
    -- May be used in address clauses:
    --
    --   Object: Some_Type;
    --   for Object use at Location (16#4000#);


  function Label (Name: String) return Address;
  pragma Interface (META, Label);
    --
    -- The LABEL meta-function allows a link name to be specified as address
    -- for an imported object in an address clause:
    --
    --  Object: Some_Type;
    --  for Object use at Label("OBJECT$$LINK_NAME");
    --
    -- System.Label returns Null_Address for non-literal parameters.
```

```
--
-- Unchecked relative address calculations
--
function "+" ( Left, Right :  Address ) return Address;
function Ord_Plus ( Left, Right :  Address ) return Address renames "+";
pragma Interface (Assembly, Ord_Plus);
pragma Interface_Information (Ord_Plus, "Ord_Plus", "", "g0,g1,g0");

function "-" ( Left, Right :  Address ) return Address;
function Ord_Minus ( Left, Right :  Address ) return Address renames "-";
pragma Interface (Assembly, Ord_Minus);
pragma Interface_Information (Ord_Minus, "Ord_Minus", "", "g0,g1,g0");

function "*" ( Left, Right :  Address ) return Address;
function Ord_Multiply ( Left, Right :  Address ) return Address renames "*";
pragma Interface (Assembly, Ord_Multiply);
pragma Interface_Information (Ord_Multiply, "Ord_Multiply", "", "g0,g1,g0");

function "/" ( Left, Right :  Address ) return Address;
function Ord_Divide ( Left, Right :  Address ) return Address renames "/";
pragma Interface (Assembly, Ord_Divide);
pragma Interface_Information (Ord_Divide, "Ord_Divide", "", "g0,g1,g0");



--
-- For the 80960MC, we choose not to include the ERROR REPORTING SUPPORT
-- (procedure Report_Error) in System.  Due to the embedded nature of the
-- 80960MC target, an unhandled exception traceback procedure is not
-- necessary in the System package.
--
--  --=========================
--  -- ERROR REPORTING SUPPORT
--  --=========================
--
--  procedure Report_Error;
--  pragma Interface (Assembly, Report_Error);
--  pragma Interface_Information (Report_Error, "REPORT_ERROR");
--
--
--  --
--  -- Report_Error can only be called in an exception handler and provides
--  --   an exception traceback like tracebacks provided for unhandled
--  --   exceptions
--  --



  --=================
  -- ORDINAL SUPPORT
  --=================

  -- The basic ordinal types are imported here to simplify use of these
```

```
  -- types.  Note: literal assignment to these types is still limited to
  -- literals in the integer range.

  type Short_Short_Ordinal is new Ordinals.Short_Short_Ordinal;
  type Short_Ordinal is new Ordinals.Short_Ordinal;
  type Ordinal is new Ordinals.Ordinal;
  type Long_Ordinal is new Ordinals.Long_Ordinal;

  function Convert is new Unchecked_Conversion (Ordinal, Address);
  function Convert is new Unchecked_Conversion (Address, Ordinal);


  --=============
  -- CALL SUPPORT
  --=============

--
   type Subprogram_Entry is
   record
     OFFSET      : Ordinals.Ordinal;
     DOMAIN_AD   : Mixed_Word;
   end record;

   type Subprogram_Value is
   record
      Proc_addr     : Address;
      Parent_frame : Address;
   end record;

    --
    -- Value returned by the implementation-defined 'Subprogram_Value
    -- attribute. The attribute is not defined for subprograms with
    -- parameters, or functions.


--
-- For the 80960MC, we choose not to include the 88K/Mips CALL procedure
-- at this time.
--
--  procedure Call (Subprogram: Subprogram_Value);
--  procedure Call (Subprogram: Address);
--
--  pragma Interface (META, Call);
--    --
--    -- The CALL meta-function allows indirect calls to subprograms
--    -- given their subprogram value. The result of a call to a nested
--    -- procedure whose parent frame does not exist (has been deallocated)
--    -- at the time of the call, is undefined.
--    --
--    -- The second form allows calls to a subprogram given its address,
--    -- as returned by the 'Address attribute. The call is undefined if
--    -- the subprogram is not a parameterless non-nested procedure.
--
```

```
--
-- For the 80960MC, we choose not to include the following 88K/Mips
-- declarations at this time.
--
--  Max_Object_Size    : CONSTANT := Max_Int;
--  Max_Record_Count   : CONSTANT := Max_Int;
--  Max_Text_Io_Count  : CONSTANT := Max_Int-1;
--  Max_Text_Io_Field  : CONSTANT := 1000;


private

   type Memory is
   record
     null;
   end record;

   --### type Mixed_Word is array(natural range 0..31) of boolean;
-- There's one more bit in Mixed_Word, but it's invisible.
   --### pragma pack(Mixed_Word);
-- for Mixed_Word'SIZE use 32;

end System;
```

System.Label

The System.Label meta-function is provided to allow you to address objects by a linker-recognized label name. This function takes a single string literal as a parameter and returns a value of System.Address. The function simply returns the run-time address of the appropriate resolved link name, the primary purpose being to address objects created and referenced from other languages.

- When used in an address clause, System.Label indicates that the Ada object or subprogram is to be referenced by a label name. The actual object must be created in some other unit (normally by another language), and this capability simply allows you to import that object and reference it in Ada. Any explicit or default initialization will be applied to the object. For example, if the object is declared to be of an access type, it will be initialized to NULL.

- When used in an expression, System.Label provides the link time address of any name, such as a name for an object or a subprogram.

5. Restrictions on representation clauses

Representation clauses are fully supported with the following exceptions:

- Enumeration representation clauses are supported for all enumeration types except Boolean types.
- Record representation clauses are supported except for records with dynamically-sized components.
- Pragma Pack is supported except for dynamically-sized components.

6. Implementation-generated names

i960 ACS has no implementation-generated names.

7. Address clause expression interpretation

An expression that appears in an object address clause is interpreted as the address of the first storage unit of the object.

8. Restrictions on unchecked conversions

Unchecked programming is supported except for unchecked type conversions where the destination type is an unconstrained record or array type.

9. Implementation-dependent characteristics of the I/O packages

Text_IO has the following implementation-dependent characteristics:

```
type Count is range 0..(2 ** 31)-2;

subtype Field is integer range 0..1000;
```

The standard run-time sublibrary contains preinstantiated versions of Text_IO.Integer_IO for types Short_Short_Integer, Short_Integer, and Integer, and of Text_IO.Float_IO for types Float, Long_Float, and Long_Long_Float. Use the following packages to eliminate multiple instantiations of the Text_IO packages:

Short_Short_Integer_Text_IO
Short_Integer_Text_IO
Integer_Text_IO
Float_Text_IO
Long_Float_Text_IO
Long_Long_Float_Text_IO