

AD-A242 288

REPOF



GE

Form Approved
OPM No. 0704-0188

2

Public reporting burden for this collection
needed, and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington
Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of
Management and Budget, Washington, DC 20503.

reviewing instructions, searching existing data sources gathering and maintaining the data
needed, and reviewing the collection of information, including suggestions for reducing this burden, to Washington
Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of
Management and Budget, Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)		2. REPORT DATE		3. REPORT TYPE AND DATES COVERED Final: 11 Apr 1991 to 01 Jun 1993	
4. TITLE AND SUBTITLE Hewlett-Packard Co./Apollo Systems Division, Domain Ada V6.0m, DN4500, Domain/OS SR10.3 (Host & Target), 910411W1.11137				5. FUNDING NUMBERS	
6. AUTHOR(S) Wright-Patterson AFB, Dayton, OH USA				7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Ada Validation Facility, Language Control Facility ASD/SCEL Bldg. 676, Rm 135 Wright-Patterson AFB, Dayton, OH 45433	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ada Joint Program Office United States Department of Defense Pentagon, Rm 3E114 Washington, D.C. 20301-3081				8. PERFORMING ORGANIZATION REPORT NUMBER AVF-VSR-451-0491	
11. SUPPLEMENTARY NOTES <i>No software available for distribution per Michelle Koe, ADA 114191 follow up</i>					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
13. ABSTRACT (Maximum 200 words) Hewlett-Packard Co./Apollo Systems Division, Domain Ada V6.0m, Wright-Patterson AFB, OH, DN4500, Domain/OS SR10.3 (Host & Target), ACVC 1.11.					
<div style="text-align: right; font-size: 2em; font-weight: bold;">91-15048</div>					
14. SUBJECT TERMS Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.				15. NUMBER OF PAGES	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED				16. PRICE CODE	
18. SECURITY CLASSIFICATION UNCLASSIFIED		19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED		20. LIMITATION OF ABSTRACT	

21-AUG-1991

90-07-05-APO

Ada COMPILER

VALIDATION SUMMARY REPORT:

Certificate Number: 910411W1.11137

Hewlett-Packard Co./Apollo Systems Division

Domain Ada V6.0m

DN4500, Domain/OS SR10.3 => DN4500, Domain/OS SR10.3

Prepared By:

Ada Validation Facility

ASD/SCEĒ

Wright-Patterson AFB OH 45433-6503

Application For

... 65241

97-1-204

... ..

7-10-68

De. 1. 1. 1.

• • • • •

ten' for

7590101

A-1

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 11 April, 1991.

Compiler Name and Version: Domain Ada V6.0m

Host Computer System: DN4500 running Domain/OS SR10.3

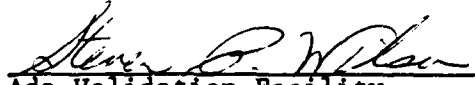
Target Computer System: DN4500 running Domain/OS SR10.3

Customer Agreement Number: 90-07-05-AP0

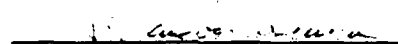
See Section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 910411W1.11137 is awarded to Hewlett-Packard Co./Apollo Systems Division. This certificate expires on 1 June 1993.


This report has been reviewed and is approved.



Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503



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



for Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

Apollo Systems Division
300 Apollo Drive
Chelmsford, Massachusetts 01824
508 256 6600
Fax 508 256 1599



Declaration of Conformance

Customer: Hewlett-Packard Co./Apollo Systems Division

Ada Validation Facility: Wright-Patterson AFB, Ohio 45433-6503

ACVC Version: 1.11

Ada Implementation:

Compiler Name and Version: Domain Ada V6.0m

Host Computer System: DN4500, Domain/OS SR10.3

Target Computer System: Same

Customer's Declaration

I, the undersigned, representing Hewlett-Packard Company, declare that Hewlett-Packard Company has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration.



Peter J. Morris
Hewlett-Packard Co.
300 Apollo Drive
Chelmsford, MA 01824

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	USE OF THIS VALIDATION SUMMARY REPORT	1-1
1.2	REFERENCES	1-2
1.3	ACVC TEST CLASSES	1-2
1.4	DEFINITION OF TERMS	1-3
CHAPTER 2	IMPLEMENTATION DEPENDENCIES	
2.1	WITHDRAWN TESTS	2-1
2.2	INAPPLICABLE TESTS	2-1
2.3	TEST MODIFICATIONS	2-4
CHAPTER 3	PROCESSING INFORMATION	
3.1	TESTING ENVIRONMENT	3-1
3.2	SUMMARY OF TEST RESULTS	3-1
3.3	TEST EXECUTION	3-2
APPENDIX A	MACRO PARAMETERS	
APPENDIX B	COMPILATION SYSTEM OPTIONS	
APPENDIX C	APPENDIX F OF THE Ada STANDARD	

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
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

INTRODUCTION

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 SPRT13, 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 SPRT13 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.

INTRODUCTION

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 AV0. The rationale for withdrawing each test is available from either the AV0 or the AVF. The publication date for this list of withdrawn tests is 14 March 1991.

E28005C	B28006C	C34006D	C35508I	C35508J	C35508M
C35508N	C35702A	C35702B	B41308B	C43004A	C45114A
C45346A	C45612A	C45612B	C45612C	C45651A	C46022A
B49008A	A74006A	C74308A	B83022B	B83022H	B83025B
B83025D	C83026A	B83026B	C83041A	B85001L	C86001F
C94021A	C97116A	C98003B	BA2011A	CB7001A	CB7001B
CB7C04A	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.

IMPLEMENTATION DEPENDENCIES

The following 201 tests have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

The following 20 tests check for the predefined type `LONG_INTEGER`:

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

C35713C, B86001U, and C86006G check for the predefined type `LONG_FLOAT`.

C35713D and B86001Z check for a predefined floating-point type with a name other than `FLOAT`, `LONG_FLOAT`, or `SHORT_FLOAT`.

A35801E checks that `FLOAT'FIRST..FLOAT'LAST` may be used as a range constraint in a floating-point type declaration; for this implementation, that range exceeds the safe numbers and must be rejected. (See section 2.3)

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point operations for types that require a `SYSTEM.MAX_MANTISSA` of 47 or greater.

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

B86001Y checks for a predefined fixed-point type other than `DURATION`.

C96005B checks for values of type `DURATION'BASE` that are outside the range of `DURATION`. There are no such values for this implementation.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A840 use representation clauses specifying non-default sizes for access types.

IMPLEMENTATION DEPENDENCIES

The tests listed in the following table are not applicable because the given file operations are supported for the given combination of mode and file access method.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT FILE	DIRECT_IO
CE2102I	CREATE	IN FILE	DIRECT_IO
CE2102J	CREATE	OUT FILE	DIRECT_IO
CE2102N	OPEN	IN FILE	SEQUENTIAL_IO
CE2102O	RESET	IN FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT FILE	DIRECT_IO
CE2102S	RESET	INOUT FILE	DIRECT_IO
CE2102T	OPEN	IN FILE	DIRECT_IO
CE2102U	RESET	IN FILE	DIRECT_IO
CE2102V	OPEN	OUT FILE	DIRECT_IO
CE2102W	RESET	OUT FILE	DIRECT_IO
CE3102E	CREATE	IN FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	-----	TEXT_IO
CE3102I	CREATE	OUT FILE	TEXT_IO
CE3102J	OPEN	IN FILE	TEXT_IO
CE3102K	OPEN	OUT FILE	TEXT_IO

CE2203A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for SEQUENTIAL_IO. This implementation does not restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for DIRECT_IO. This implementation does not restrict file capacity.

CE3304A checks that USE_ERROR is raised if a call to SET LINE LENGTH or SET PAGE LENGTH specifies a value that is inappropriate for the external file. This implementation does not have inappropriate values for either line length or page length.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST. For this implementation, the value of COUNT'LAST is greater than 150000, thus making the checking of this objective impractical.

IMPLEMENTATION DEPENDENCIES

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 22 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.

B20049A	B33301B	B38003A	B38003B	B38009A	B38009B
B85008G	B85008H	BC1303F	BC3005B	BD2B03A	BD2D03A
BD4003A					

A35801E was graded inapplicable by Evaluation Modification as directed by the AVO; the compiler rejects the use of the range `FLOAT'FIRST..FLOAT'LAST` as the range constraint of a floating-point type declaration because the bounds lie outside of the range of safe numbers (cf. ARM 3.5.7(12)).

CD1009A, CD1009I, CD1C03A, CD2A22J, CD2A24A, and CD2A31A..C (3 tests) were graded passed by Evaluation Modification as directed by the AVO. These tests use instantiations of the support procedure `LENGTH_CHECK`, which uses `Unchecked_Conversion` according to the interpretation given in AI-00590. The AVO ruled that this interpretation is not binding under ACVC 1.11; the tests are ruled to be passed if they produce Failed messages only from the instances of `LENGTH_CHECK`--i.e, the allowed Report.Failed messages have the general form:

" * CHECK ON REPRESENTATION FOR <TYPE_ID> FAILED."

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 a point of contact for technical information about this Ada implementation system, see:

Peter J. Morris
Hewlett-Packard Co
300 Apollo Drive
Chelmsford MA 01824

For a point of contact for sales information about this Ada implementation system, see:

Peter J. Morris
Hewlett-Packard Co
300 Apollo Drive
Chelmsford MA 01824

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.

PROCESSING INFORMATION

a) Total Number of Applicable Tests	3807
b) Total Number of Withdrawn Tests	93
c) Processed Inapplicable Tests	69
d) Non-Processed I/O Tests	0
e) Non-Processed Floating-Point Precision Tests	201
f) Total Number of Inapplicable Tests	270
g) Total Number of Tests for ACVC 1.11	4170

All I/O tests of the test suite were processed because this implementation supports a file system. The above number of floating-point tests were not processed because they used floating-point precision exceeding that supported by the implementation. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 270 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. In addition, the modified tests mentioned in section 2.3 were also processed.

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 read by two workstations, in an adjacent building. The contents were then copied over the network to the host computers.

Testing was done on three host computers connected together on a network. After the test files were loaded onto the host computers, the full set of tests was processed by the Ada implementation.

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 during this test were:

Option Switch	Effect

-el	Produce Error Listing

PROCESSING INFORMATION

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

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
\$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	'"' & (1..V-2 => 'A') & '"'

MACRO PARAMETERS

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

Macro Parameter	Macro Value
\$MAX_IN_LEN	499
\$ACC_SIZE	32
\$ALIGNMENT	4
\$COUNT_LAST	2147483647
\$DEFAULT_MEM_SIZE	16777216
\$DEFAULT_STOR_UNIT	8
\$DEFAULT_SYS_NAME	APOLLO_UNIX_M68K
\$DELTA_DOC	0.0000000004656612873077392578125
\$ENTRY_ADDRESS	SYSTEM. "+"(16#40#)
\$ENTRY_ADDRESS1	SYSTEM. "+"(16#80#)
\$ENTRY_ADDRESS2	SYSTEM. "+"(16#100#)
\$FIELD_LAST	2147483647
\$FILE_TERMINATOR	' '
\$FIXED_NAME	NO_SUCH_FIXED_TYPE
\$FLOAT_NAME	NO_SUCH_FLOAT_NAME
\$FORM_STRING	""
\$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
\$GREATER_THAN_DURATION	100_000.0
\$GREATER_THAN_DURATION BASE LAST	10_000_000.0
\$GREATER_THAN_FLOAT_BASE LAST	1.8E+308
\$GREATER_THAN_FLOAT_SAFE LARGE	1.0E308

MACRO PARAMETERS

```

$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
    1.0E308

$HIGH_PRIORITY      99

$ILLEGAL_EXTERNAL_FILE_NAME1
    7illegal/file_name/2{J$X2102C.DAT

$ILLEGAL_EXTERNAL_FILE_NAME2
    7illegal/file_name/CE2102C*.DAT

$INAPPROPRIATE_LINE_LENGTH
    -1

$INAPPROPRIATE_PAGE_LENGTH
    -1

$INCLUDE_PRAGMA1    PRAGMA INCLUDE ("A28006D1.A")
$INCLUDE_PRAGMA2    PRAGMA INCLUDE ("B28006F1.A")

$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
    -10_000_000.0

$LINE_TERMINATOR    ASCII.LF

$LOW_PRIORITY       0

$MACHINE_CODE_STATEMENT
    CODE_2'(MOVEA_L,A0,A5);

$MACHINE_CODE_TYPE   OPCODE

$MANTISSA_DOC        31

$MAX_DIGITS          15

$MAX_INT             2147483647
$MAX_INT_PLUS_1      2147483648
$MIN_INT             -2147483648

```

MACRO PARAMETERS

\$NAME	TINY_INTEGER
\$NAME_LIST	APOLLO_UNIX_A88K,APOLLO_UNIX_M68K
\$NAME_SPECIFICATION1	//spas/acvc.m/results/ctests/ce/ ce2120a.lib/X2120A
\$NAME_SPECIFICATION2	//spas/acvc.m/results/ctests/ce/ ce2120b.lib/X2120B
\$NAME_SPECIFICATION3	//spas/acvc.m/results/ctests/ce/ ce3119a.lib/X3119A
\$NEG_BASED_INT	16#F0000000E#
\$NEW_MEM_SIZE	16_777_216
\$NEW_STOR_UNIT	8
\$NEW_SYS_NAME	APOLLO_UNIX_A88K
\$PAGE_TERMINATOR	ASCII.LF&ASCII.FF
\$RECORD_DEFINITION	TYPE CODE 0 (OP:OPCODE) IS RECORD NULL; END RECORD;
\$RECORD_NAME	CODE_0
\$TASK_SIZE	32
\$TASK_STORAGE_SIZE	2000
\$TICK	0.01
\$VARIABLE_ADDRESS	VAR_1'ADDRESS
\$VARIABLE_ADDRESS1	VAR_2'ADDRESS
\$VARIABLE_ADDRESS2	VAR_3'ADDRESS
\$YOUR_PRAGMA	EXTERNAL_NAME

APPENDIX B

COMPILATION SYSTEM OPTIONS

COMPILER OPTIONS

The compiler 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.

`-# id type value (define)` Define an identifier of a specified type and value.

`-a file_name (archive)` Treat `file_name` as an ar file. Since some archive files end with `.a`, use `-a` to distinguish archive files from Ada source files.

`-d (dependencies)` Analyze for dependencies only. Do not do semantic analysis or code generation. Update the library, marking any defined units as uncompiled. `a.make` uses the `-d` option to establish dependencies among new files.

`-e (error)` Process compilation error messages using `a.error` and send it to stdout (standard output). Only the source lines containing errors are listed. Use only one `-e` or `-E` option.

`-E`

`-E file`

`-E directory (error output)` Without a file or directory argument, `ada` processes error messages using `a.error` and directs a brief output to stdout; the raw error messages are left in `ada_source.err`. If a file pathname is specified, the raw error messages are placed in that file. If a directory argument is supplied, the raw error output is placed in `dir/source.err`. Then, the file of raw error messages can be used as input to `a.error`. Use only one `-e` or `-E` option.

COMPILATION SYSTEM OPTIONS

-el (error listing) Intersperse error messages among source lines and direct to stdout.

-El

-El file

-El directory (error listing) Same as the **-E** option, except that a source listing with errors is produced.

-ev (error vi) Process syntax error messages using a.error, embed them in the source file, and call the environment editor, EDITOR.

-K (keep) Keep the Intermediate Language (IL) file produced by the compiler front end. The IL file will be placed in the .objects directory, with the filename ada_source.i If the **-O** option is used, the compiler places an ada_source.O file in the same directory.

-L ada_library (library) Operate in Domain/Ada library library_name (the current working directory is the default).

-lfile abbreviation (library) This is an option passed to the Domain/OS linker telling it to search the specified library file. (Do not put a space between the **-l** and the file abbreviation.)

-M unit_name (main) Produce an executable program using the named unit as the main program. The unit must be either a parameterless procedure or a parameterless function returning an integer. The executable program will be left in the file a.out unless overridden with the **-o** option.

-M ada_source.a (main) Similar to **-M unit_name**, except that the unit name is assumed to be the root name of the .a file (In the example **-M example.a**, the unit name is assumed to be example). Only one .a file can be preceded by **-M**.

-n Suppress the generation of symbol table information (for use by Apollo's performance analysis tools, tb and dpat) in the object module.

-o executable_file (output) Use this option in conjunction with the **-M** option. executable_file is the name of the executable rather than the default, a.out.

COMPILATION SYSTEM OPTIONS

-O[0-9] (optimize) Invoke the code optimizer (no space before the digit). An optional digit limits the number of passes by the optimizer; without the **-O** option, four passes are made (default). The option levels are:

- O** Full optimization
- O0** Prevents optimization
- O1** No hoisting
- O2** No hoisting, but more passes
- O3** No hoisting, but even more passes
- O4** Hoisting from loops
- O5** Hoisting from loops, but more passes
- O6** Hoisting from loops with maximum passes
- O7** Hoisting from loops and branches
- O8** Hoisting from loops and branches, more passes
- O9** Hoisting from loops and branches, maximum passes

-P (preprocessor) Invoke the Domain/Ada preprocessor. See Chapter 6 for a detailed discussion.

-R Domain/Ada library (recompile instantiation) Force analysis of all generic instantiations, causing reinstantiation of any that are out of date.

-S (suppress) Apply pragma **SUPPRESS** to the entire compilation for all suppressible checks.

-sh (show) Display the pathnames of the compiler components.

-T (timing) Print timing information for the compilation.

-v (verbose) Print compiler version number, date and time of compilation, name of file compiled, command input line, total compilation time, and error summary line.

-w (warnings) Suppress warning diagnostics.

COMPILATION SYSTEM OPTIONS

LINKER OPTIONS

The 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 linker documentation and not to this report.

- E unit_name (elaborate) Elaborate unit_name as early in the elaboration order as possible.
- F (files) Print a list of dependent files in order and suppress linking.
- o executable_file (output) Use the specified filename as the name of the output rather than the default, a.out.
- sh (show) Display the pathname of the tool executable but do not execute it.
- U (units) Print a list of dependent units in order and suppress linking.
- v (verbose) Print the linker command before executing it.
- V (verify) Print the linker command, but suppress execution.

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:

package STANDARD is

```
...
type INTEGER is range -2147483648 .. 2147483647;
type SHORT_INTEGER is range -32768 .. 32767;
type TINY_INTEGER is range -128 .. 127;

type FLOAT is digits 15 range -1.79769313486231E+308 ..
                                     1.79769313486231E+308;
type SHORT_FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;

type DURATION is delta 1.000000000000000E-3 range -2147483.648 ..
                                                    2147483.647;
...
```

end STANDARD;

MC680X0 Implementation-Dependent Characteristics

This section describes the Domain/Ada MC680X0 as required by Appendix F of the Ada Language Reference Manual (RM).

Implementation-Dependent Pragmas and Attributes

This subsection details the Domain/Ada MC680X0 implementation-dependent pragmas and attributes.

Implementation-Dependent Pragmas

Domain/Ada provides the following pragmas in its MC680X0 implementation:

BUILT_IN

Pragma BUILT_IN is used in the implementation of some predefined Ada packages, but provides no user access. It is used only to implement code bodies for which no actual Ada body can be provided, for example the MACHINE_CODE package.

EXTERNAL_NAME

Pragma EXTERNAL_NAME allows a link_name for an Ada variable or subprogram to be specified so that the object from programs written in other languages can be referenced.

IMPLICIT_CODE

Pragma IMPLICIT_CODE specifies that implicit code generated by the compiler is allowed (ON) or disallowed (OFF). This pragma can be used to be specified only within the declarative part of a machine code procedure.

INLINE_ONLY

Pragma INLINE_ONLY, when used in the same way as pragma INLINE, indicates to the compiler that the subprogram must always be inlined. (This is very important for some code procedures.) This pragma also suppresses the generation of a callable version of the routine, which saves code space.

INTERFACE_NAME

Pragma INTERFACE_NAME allows variables and subprograms defined in another language to be referenced from Ada programs, replacing all occurrences of a ada_subprogram_or_object_name with an external

reference to a link name in the object file. The pragma `INTERFACE_NAME`, used in conjunction with pragma `INTERFACE`, allows the exact name of the subprogram being called to be specified by providing an optional linker name for the subprogram. This optional linker name enables the calling of a subprogram defined in another language whose name contains characters that are not allowed in an Ada identifier.

`NO_IMAGE`

Pragma `NO_IMAGE` suppresses the generation of the image array used for the `IMAGE` attribute of enumeration types. This eliminates the overhead required to store the array in the executable image.

`NON_REENTRANT`

Pragma `NON_REENTRANT` takes one argument which can be the name of either a library subprogram or a subprogram declared immediately within a library package spec or body. It indicates to the compiler that the subprogram will not be called recursively, thus allowing the compiler to perform specific optimizations. The pragma can be applied to a subprogram or a set of overloaded subprograms within a package spec or package body.

`NOT_ELABORATED`

Pragma `NOT_ELABORATED`, which is allowed only within a package specification, suppresses elaboration checks for all entities defined within a package, including the package specification itself. In addition, this pragma suppresses the generation of elaboration code. When using pragma `NOT_ELABORATED`, there must be no entities defined in the program that require elaboration.

`OPTIMIZE_CODE`

Pragma `OPTIMIZE_CODE` takes one of the identifiers `ON` or `OFF` as the single argument. This pragma is only allowed within a machine code procedure. It specifies whether the code should be optimized by the compiler. The default is `ON`. When `OFF` is specified, the compiler will generate the code as specified.

`PASSIVE`

Pragma `PASSIVE` has three forms:

```
PRAGMA PASSIVE;
PRAGMA PASSIVE(SEMAPHORE);
PRAGMA PASSIVE(INTERRUPT, <number>);
```

This pragma can be applied to a task or task type declared immediately within a library package spec or body. The pragma directs the compiler to optimize certain tasking operations. It is possible that the statements in a task body will prevent the intended optimization;

APPENDIX F OF THE Ada STANDARD

in these cases, a warning will be generated at compile time and will raise `TASKING_ERROR` at runtime.

`SHARE_CODE`

Pragma `SHARE_CODE` provides for the sharing of object code between multiple instantiations of the same generic procedure or package body. A "parent" instantiation is created, and subsequent instantiations of the same types can share the parent's object code, reducing program size and compilation times. The name pragma `SHARE_BODY` instead of `SHARE_CODE` can be used with the same effect.

In addition to the pragmas mentioned in the previous list, the Domain/Ada MC680X0 implementation expands upon the functionality of the following predefined language pragmas:

`INLINE`

Pragma `INLINE` is implemented as described in Appendix B of the RM with the addition that recursive calls can be expanded up to the maximum depth of 8. The compiler produces warnings for nestings that are too deep or for bodies that are not available for inline expansion.

`PACK`

Pragma `PACK` causes the compiler to minimize gaps between components in the representation of composite types. For arrays, the compiler packs components to bit sizes corresponding to powers of 2 (if the field is smaller than `STORAGE_UNIT` bits). The compiler packs objects larger than a single `STORAGE_UNIT` to the nearest `STORAGE_UNIT`.

`SUPPRESS`

Pragma `SUPPRESS` is supported in the single parameter form. The pragma applies from the point of occurrence to the end of the innermost enclosing block. `DIVISION_CHECK` cannot be suppressed. The double parameter form of the pragma with a name of an object, type, or subtype is recognized, but has no effect in the current release. This pragma can be used to suppress elaboration checks on any compilation unit except a package specification.

The Domain/Ada MC680X0 implementation recognizes the following pragmas, but they have no effect in the current release:

Pragma CONTROLLED

Pragma MEMORY_SIZE

Pragma OPTIMIZE

Pragma SHARED

Pragma STORAGE UNIT (This implementation does not allow modification of package SYSTEM by means of pragmas. However, the same effect can be achieved by recompiling package SYSTEM with altered values.)

Pragma SYSTEM NAME (This implementation does not allow modification of package SYSTEM by means of pragmas. However, the file system.a can be copied from the STANDARD library to a local Domain/Ada library and recompiled there with the new values.)

The following pragmas are implemented as described in Appendix B of the RM:

Pragma ELABORATE

Pragma INTERFACE (supports C and FORTRAN only)

Pragma LIST

Pragma PAGE

Pragma PRIORITY

Implementation-Defined Attribute: 'REF

The Domain/Ada MC680X0 implementation provides one implementation-defined attribute, 'REF. Attribute 'REF can be used in one of two ways: X'REF and SYSTEM.ADDRESS'REF(N). X'REF can be used only in machine code procedures SYSTEM.ADDRESS'REF(N) can be used anywhere that an integer expression is to be converted to an address.

X'REF

The X'REF attribute generates a reference to the entity to which it is applied.

In X'REF, X must be either a constant, variable, procedure, function, or label. The attribute returns a value of the type MACHINE_CODE.OPERAND, which can be used only to designate an operand within a machine code-statement.

The instruction generated by the code-statement in which the attribute occurs can be preceded by additional instructions needed to facilitate the reference (for example, loading a base register). If the declarative section of the procedure contains pragma `IMPLICIT_CODE (OFF)`, the compiler will generate a warning if additional code is required.

References can also cause the generation of run-time checks. Pragma `SUPPRESS` can be used to eliminate these checks.

```
CODE_1'(JSR, PROC'REF);
CODE_2'(MOVE_L, X.ALL(Z)'REF, DO);
```

SYSTEM.ADDRESS'REF(N)

The effect of `SYSTEM.ADDRESS'REF(N)` is similar to the effect of an unchecked conversion from integer to address. However, this attribute should be used instead of an unchecked conversion in the following circumstances (in these circumstances, N must be static):

Within any of the run-time configuration packages:

Use of unchecked conversion within an address clause would require the generation of elaboration code, but the configuration packages are not elaborated.

In any instance where N is greater than `INTEGER'LAST`:

Such values are required in address clauses that reference the upper portion of memory. To use unchecked conversion in these instances would require that the expression be given as a negative integer.

To place an object at an address, use the `'REF` attribute:

The `integer_value`, in the following example, is converted to an address for use in the address clause representation specification. The form avoids `UNCHECKED_CONVERSION` and is also useful for 32-bit unsigned addresses.

```
--place an object at an address
for object use at ADDRESS'REF (integer_value)

--to use unsigned addresses
for VECTOR use at SYSTEM.ADDRESS'REF(16#808000d0#);
TOP_OF_MEMORY: SYSTEM.ADDRESS:= SYSTEM.ADDRESS'REF(16#FFFFFFFF#);
```

In SYSTEM.ADDRESS'REF(N), SYSTEM.ADDRESS must be the type SYSTEM.ADDRESS. N must be an expression of type UNIVERSAL_INTEGER. The attribute returns a value of type SYSTEM.ADDRESS, which represents the address designated by N.

Specification of the Package SYSTEM

with UNSIGNED_TYPES;
package SYSTEM is

```
pragma SUPPRESS(ALL_CHECKS);
pragma SUPPRESS(EXCEPTION_TABLES);
pragma NOT_ELABORATED;
```

```
type NAME is ( apollo_unix_a88k, apollo_unix_m68k );
```

```
SYSTEM_NAME          : constant NAME := apollo_unix_m68k;
```

```
STORAGE_UNIT         : constant := 8;
```

```
MEMORY_SIZE          : constant := 16_777_216;
```

-- System-Dependent Named Numbers

```
MIN_INT               : constant := -2_147_483_648;
```

```
MAX_INT               : constant := 2_147_483_647;
```

```
MAX_DIGITS            : constant := 15;
```

```
MAX_MANTISSA          : constant := 31;
```

```
FINE_DELTA            : constant := 2.0*(-31);
```

```
TICK                  : constant := 0.01;
```

-- Other System-dependent Declarations

```
subtype PRIORITY is INTEGER range 0 .. 99;
```

```
MAX_REC_SIZE : integer := 64*1024;
```

```
type ADDRESS is private;
```

```
function ">" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
```

```
function "<" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
```

```
function ">=" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
```

```
function "<=" (A: ADDRESS; B: ADDRESS) return BOOLEAN;
```

```
function "-" (A: ADDRESS; B: ADDRESS) return INTEGER;
```

```
function "+" (A: ADDRESS; I: INTEGER) return ADDRESS;
```

```
function "-" (A: ADDRESS; I: INTEGER) return ADDRESS;
```

```
function "+" (I: UNSIGNED_TYPES.UNSIGNED_INTEGER) return ADDRESS;
```

```
function MEMORY_ADDRESS
```

```
  (I: UNSIGNED_TYPES.UNSIGNED_INTEGER) return ADDRESS renames "+";
```

```
NO_ADDR : constant ADDRESS;
```

APPENDIX F OF THE Ada STANDARD

```
type TASK_ID is private;
NO_TASK_ID : constant TASK_ID;

type PROGRAM_ID is private;
NO_PROGRAM_ID : constant PROGRAM_ID;

private

type ADDRESS is new UNSIGNED_TYPES.UNSIGNED_INTEGER;

NO_ADDR : constant ADDRESS := 0;

pragma BUILT_IN(">");
pragma BUILT_IN("<");
pragma BUILT_IN(">=");
pragma BUILT_IN("<=");
pragma BUILT_IN("-");
pragma BUILT_IN("+");

type TASK_ID is new UNSIGNED_TYPES.UNSIGNED_INTEGER;
NO_TASK_ID : constant TASK_ID := 0;

type PROGRAM_ID is new UNSIGNED_TYPES.UNSIGNED_INTEGER;
NO_PROGRAM_ID : constant PROGRAM_ID := 0;

end SYSTEM;
```

Restrictions on Representation Clauses and Unchecked Type Conversions

This section summarizes the restrictions on representation clauses and unchecked type conversions for the MC680X0 implementation of Domain/Ada.

Representation Clauses

The Domain/Ada MC680X0 implementation supports bit level, length, enumeration, size, and record representation clauses. Size clauses are not supported for tasks, floating-point types, or access types. This implementation supports address clauses for objects except for task objects and for initialized objects given dynamic addresses. Address clauses for task entries are supported; the specified value is a UNIX signal value.

The only restrictions on record representation clauses are the following:

If a component does not start and end on a storage unit boundary, it must be possible to get the component into a register with one move instruction. On a MC680X0 machine, where longwords start on even bytes, the component must fit into 4 bytes starting on a word boundary.

A component that is itself a record must occupy a number of bits equal to a power of two. Components that are of a discrete type or packed array can occupy an arbitrary number of bits subject to the previously mentioned restrictions.

Unchecked Type Conversions

This implementation of Domain/Ada supports the generic function `UNCHECKED_CONVERSION` with the following restriction:

The predefined generic function `UNCHECKED_CONVERSION` cannot be instantiated with a target type that is an unconstrained array type or an unconstrained record type with discriminants.

Denoting Implementation-Dependent Components in Record Representation Clauses

Record representation clauses are based on the target machine's word, byte, and bit order numbering so that Domain/Ada is consistent with various machine architecture manuals. Bits within a `STORAGE_UNIT` are also numbered according to the target machine manuals. This implementation of Domain/Ada does not support the allocation of implementation-dependent components in records.

Interpretations of Expressions in Address Clauses

This implementation of Domain/Ada supports the `SYSTEM.ADDRESS'REF(N)` summarized in Section F.1.2.2.

Implementation-Dependent Characteristics of I/O Packages

The Ada I/O system is implemented using Domain/OS I/O. Both formatted I/O and binary I/O are available. There are no restrictions on the types with which `DIRECT_IO` and `SEQUENTIAL_IO` can be instantiated except that the element size must be less than a maximum given by the variable `SYSTEM.MAX_REC_SIZE`. This variable can be set to any value prior to the generic instantiation; thus, any element size can be used. `DIRECT_IO` can be instantiated with unconstrained types, but each element will be padded out to the maximum possible for that type or to `SYSTEM.MAX_REC_SIZE`, whichever is smaller. No checking other than normal static Ada type checking is done to ensure that values from files are read into correctly sized and typed objects.

Domain/Ada file and terminal input/output are identical in most respects and differ only in the frequency of buffer flushing. Output is buffered (buffer size is 1024 bytes), and the buffer is flushed after each write request if the destination is a terminal.

APPENDIX F OF THE Ada STANDARD

The procedure `FILE_SUPPORT.ALWAYS_FLUSH` (`file_ptr`) will cause flushing of the buffer associated with `file_ptr` after all subsequent output requests. Refer to the source code for `file_sprpt.o.a` in the standard library for more information.

Instantiations of `DIRECT_IO`

Instantiations of `DIRECT_IO` use the value `MAX_REC_SIZE` as the record size (expressed in `STORAGE_UNITS`) when the size of `ELEMENT_TYPE` exceeds that value. For example, for unconstrained arrays such as a string where `ELEMENT_TYPE'SIZE` is very large, `MAX_REC_SIZE` is used instead. `MAX_REC_SIZE` (defined in package `SYSTEM`) can be changed before instantiating `DIRECT_IO` to provide an upper limit on the record size. The maximum size supported is $1024 * 1024 * \text{STORAGE_UNIT bits}$. `DIRECT_IO` will raise `USE_ERROR` if `MAX_REC_SIZE` exceeds this absolute limit.

Instantiations of `SEQUENTIAL_IO`

Instantiations of `SEQUENTIAL_IO` use the value `MAX_REC_SIZE` as the record size (expressed in `STORAGE_UNITS`) when the size of `ELEMENT_TYPE` exceeds that value. For example, for unconstrained arrays such as `STRING` where `ELEMENT_TYPE'SIZE` is very large, `MAX_REC_SIZE` is used instead. `MAX_REC_SIZE` (defined in package `SYSTEM`) can be changed before instantiating `SEQUENTIAL_IO` to provide an upper limit on the record size. `SEQUENTIAL_IO` imposes no limit on `MAX_REC_SIZE`.

Additional Implementation-Dependent Features

This section details any other features that are specific to the Domain/Ada MC680X0 implementation.

Restrictions on "Main" Programs

Domain/Ada requires that a "main" program must be a non-generic subprogram that is either a procedure or a function returning an `Ada STANDARD.INTEGER` (the predefined type). In addition, a "main" program cannot be an instantiation of a generic subprogram.

Generic Declarations

Domain/Ada does not require that a generic declaration and the corresponding body be part of the same compilation, and they are not required to exist in the same Domain/Ada library. The compiler generates an error if a single compilation contains two versions of the same unit.

Implementation-Dependent Portions of Predefined Ada Packages

Domain/Ada supplies the following predefined Ada packages given by the Ada RM C(22) in the standard library:

```
package STANDARD

package CALENDAR

package SYSTEM

generic procedure UNCHECKED_DEALLOCATION

generic function UNCHECKED_CONVERSION

generic package SEQUENTIAL_IO

generic package DIRECT_IO

package TEXT_IO

package IO_EXCEPTIONS

package LOW_LEVEL_IO

package MACHINE_CODE
```

The implementation-dependent portions of the predefined Ada packages define the following types and objects:

<in package STANDARD>

```
type BOOLEAN is          <8-bit, byte>;
type TINY_INTEGER is     <8-bit, byte integer>;
type SHORT_INTEGER is    <16-bit, word integer>;
type INTEGER is          <32-bit, longword integer>;
type SHORT_FLOAT is      <6-digit, 32-bit, float>;
type FLOAT is            <15-digit, 64-bit, float>;
type DURATION is delta 1.000000000000000E-03 range
                        -2147483.648 .. 2147483.647;
```

<in package DIRECT_IO>

```
type COUNT is range 0 .. 2_147_483_647;
```

<in package TEXT_IO>

```
type COUNT is range 0 .. 2_147_483_647;
subtype FIELD is INTEGER range 0 .. INTEGER'last;
```

APPENDIX F OF THE Ada STANDARD

Values of Integer Attributes

The MC680X0 implementation of Domain/Ada provides three integer types in addition to universal_integer: INTEGER, SHORT_INTEGER, and TINY_INTEGER. Table F-1 lists the ranges for these integer types.

Table F-1. Domain/Ada Integer Types

Domain/Ada Integer Types

Name of Attribute	Attribute Value of INTEGER	Attribute Value of SHORT_INTEGER	Attribute Value of TINY_INTEGER
FIRST	-2_147_483_648	-32_768	-128
LAST	2_147_483_647	32_767	127

Values of Floating-Point Attributes

Table F-2 lists the attributes of floating-point types.

Table F-2. Domain/Ada Floating-Point Types

Name of Attribute	Attribute Value of FLOAT	Attribute Value of SHORT_FLOAT
SIZE	64	32
FIRST	-1.79769313486231E+308	-3.40282E+38
LAST	1.79769313486231E+308	3.40282E+38
DIGITS	15	6
MANTISSA	51	21
EPSILON	8.88178419700125E-16	9.53674316406250E-07
EMAX	204	84
SMALL	1.94469227433160E-62	2.58493941422821E-26
LARGE	2.5710087081438E+61	1.93428038904620E+25
SAFE_EMAX	102	126
SAFE_SMALL	1.11253692925360E-308	5.87747175411143E-39
SAFE_LARGE	4.49423283715578E+307	8.5075511654154E+37
MACHINE_RADIX	2	2
MACHINE_MANTISSA	53	24
MACHINE_EMAX	1024	128
MACHINE_EMIN	-1022	-126
MACHINE_ROUNDING	TRUE	TRUE
MACHINE_OVERFLOW	TRUE	TRUE

Attributes of Type DURATION

Table F-3 lists the attributes for the fixed-point type DURATION.

Table F-3. Attributes for the Fixed-Point Type DURATION

Name of Attribute	Attribute Value for DURATION
SIZE	32
FIRST	-2147483.648
LAST	2147483.647
DELTA	1.000000000000000E-03
MANTISSA	31
SMALL	9.765625000000000E-04
LARGE	4.19430399902343E_06
FORE	8
AFT	3
SAFE_SMALL	9.765625000000000E-04
SAFE_LARGE	4.19430399902343E+06
MACHINE_ROUNDS	TRUE
MACHINE_OVERFLOW	TRUE

Implementation Limits

Character Set: Domain/Ada provides the full graphic character textual representation for programs. The character set for source files and internal character representations is ASCII.

Lexical Elements, Separators, and Delimiters: Domain/Ada uses normal Domain/OS I/O text files as input. Each line is terminated by a newline character (ASCII.LF).

Source File Limits: Domain/Ada imposes the following limitations on source files:

499 characters per source line

1296 Ada units per source file

32767 lines per source file

Compiler/Tool Limits: Domain/Ada imposes the following limits on the use of the Domain/Ada compiler:

- 499 characters in identifiers and literals
- 4,000,000 STORAGE UNITS in a statically sized record or array
- 32,768 bytes as the STORAGE_SIZE default for a task
- No limit on the number of declared objects (except virtual space)
- 800 characters in a rooted name (full pathname of an object)
- 8 recursive inlines
- 8 nested inlines
- 400 nested constructs
- 2048 characters in ADAPATH (library search list)
- 2048 characters in a WITH or INFO directive
- 16M of memory use per compilation (other Domain/OS limits may apply)
- 50 lexical errors before the front end exits
- 100 syntax errors before the front end exits
- 10 attempts to lock GVAS_table
- 10 attempts to lock ada.lib
- 20 attempts to lock gnrx.lib
- 64 debugger breakpoints
- 32 debugger array dimensions in a p command
- 9 debugger 'call parameters'
- 256 debugger 'run parameters'