ACAL Control Number: EDS19980904ACT02-2.1
DATE COMPLETED
BEFORE ON-SITE: 21 OCT 98
AFTER ON-SITE: 01 DEC 98

Ada Conformity Assessment Test Report
Certificate Number: A981103E2.1-042
Ada Core Technologies, Inc. and Electric Boat Corporation
GNATWorks Professional Ada95, Version 4.12,
SPARC Solaris cross to VxWorks / M68K family,
Sun UltraSPARC-II under Solaris, 2.5.1 =>
DY-4 SVME/DMV-163 (68040) under VxWorks, 5.3.1/Tornado

(Final)

Prepared By:
Ada Conformity Assessment Laboratory
EDS Conformance Testing Center
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton, OH 45424-0593
U.S.A.

(c) Copyright 1998, Electronic Data Systems Corporation
This document is copyrighted. It may be reproduced by any means and by any person or entity, but only in its entirety. Reproduction of any smaller part of this report is prohibited.

TABLE OF CONTENTS

ACAL Control Number: EDS19980904ACT02-2.1

PREFACE

This report documents the conformity assessment of an Ada processor. This assessment was conducted in accordance with the Ada Conformity Assessment Procedures of the Ada Conformity Assessment Laboratory (ACAL) named below and with the Ada Conformity Assessment Authority Operating Procedures, Version 1.3. The Ada Conformity Assessment Test Suite (ACATS), Version 2.1, was used for testing; The specific version identification is given below.

The successful completion of conformity assessment is the basis for the issuance of a certificate of conformity and for subsequent registration of related processors. A copy of the certificate A981103E2.1-042 which was awarded for this assessment is presented on the following page. Conformity assessment does not ensure that a processor has no nonconformities to the Ada standard other than those, if any, documented in this report. The compiler vendor declares that the tested processor contains no deliberate deviation from the Ada standard; a copy of this Declaration of Conformity is presented immediately after the certificate.

Base Test Suite Version	ACATS 2.1 (VCS label A2_1B) (See Section 2.2 for details)
Location of Testing	Ada Core Technologies, Inc. 73 Fifth Ave., Suite 11B New York NY 10003
Test Completion Date	3 November 1998

This report has been reviewed and approved by the signatories below. These organizations attest that, to the best of their knowledge, this report is accurate and complete; however, they make no warrant, express or implied, that omissions or errors have not occurred.

Ada Conformity Assessment Laboratory
Phil Brashear
EDS Conformance Testing Center
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton OH 45424-0593
U.S.A.

Ada Conformity Assessment Authority
Randall Brukardt
ACAA
P.O. Box 1512
Madison WI 53701
U.S.A.

(Insert copy of certificate here)

Results Summary for A981103E2.1-042

Specialized Needs Annexes

Note: Tests allocated to these annexes are processed only when the vendor claims support.

SPECIALIZED NEEDS ANNEXES	Total	With-Drawn	Passed	Inappli-cable	Unsup-ported
C Systems Programming & required Section 13 (representation support)	24 161 --- 185	2 1 --- 3	22 160 --- 182	0 0 --- 0	0 0 --- 0
D Real-Time Systems (which requires Annex C)	58	5	47	6	0
E Distributed Systems	26	0	17	9	0
F Information Systems	21	0	21	0	0
G Numerics	29	1	28	0	0
H Safety and Security	30	0	30	0	0

DECLARATION OF CONFORMITY

Customer: Ada Core Technologies, Inc.

Ada Conformity Assessment Laboratory: EDS Conformance Testing Center
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton OH 45424-0593
U.S.A.

ACATS Version: 2.1

Ada Processor

Ada Compiler Name and Version: GNATWorks Professional Ada95,
Version 4.12,
SPARC Solaris cross to
VxWorks / M68K family

Host Computer System: Sun UltraSPARC-II
Solaris, 2.5.1

Target Computer System: DY-4 SVME/DMV-163 (68040)
VxWorks, 5.3.1/Tornado

Declaration

I, the undersigned, declare that I have no knowledge of deliberate deviations from the Ada Language Standard ANSI/ISO/IEC 8652:1995, FIPS PUB 119-1 other than the omission of features as documented in this Conformity Assessment Summary Report.

_____ _____
Customer Signature Date

_____ _____
Certificate Awardee Signature Date

_____ _____
Certificate Awardee Signature Date

CHAPTER 1

INTRODUCTION

The Ada processor described above was tested in accordance with the Ada Conformity Assessment Procedures of the ACAL and with Version 1.3 of the Operating Procedures of the ACAA [Pro98]. Testing was accomplished using Version 2.1 of the Ada Conformity Assessment Test Suite (ACATS), also known as the Ada Compiler Validation Capability (ACVC). The ACATS checks the conformity of an Ada processor to the Ada Standard [Ada95].

This Ada Conformity Assessment Test Report (ACATR) gives an account of the testing of this Ada processor. For any technical terms used in this report, the reader is referred to [Pro98]. A detailed description of the ACATS may be found in the ACVC User's Guide [UG97].

1.1 USE OF THIS REPORT

Consistent with the national laws of the originating country, the ACAL and ACAA 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). Certified status is awarded only to the processor identified in this report. Copies of this report are available to the public from the ACAL that performed this conformity assessment.

Questions regarding this report or the test results should be directed to the ACAL which performed this conformity assessment or to the Ada Conformity Assessment Authority. For all points of contact, see Appendix B.

1.2 TEST CLASSES

Compliance of Ada processors is tested by means of the ACATS. The ACATS 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 most 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 contains constants of type SYSTEM.ADDRESS. These constants are used by selected Section 13 tests and by isolated tests for other sections. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for the Input-Output features of the Ada Standard, defined in Annex A of [Ada 95]. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, conformity 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 processor correctly detects violation of the Ada Standard involving multiple, separately compiled units. In most Class L tests, errors are expected at link time, and execution must not begin. Other L tests may execute and report the appropriate result.

For some tests of the ACATS, certain implementation-specific values must be supplied. Two insertion methods for the implementation-specific values are used: a macro substitution on the source file level of the test, and linking of a package that contains the implementation-specific values. Details are described in [UG97]. A list of the values used for this processor, along with the specification and body of the package (and children applicable to any of Specialized Needs Annexes being tested) are provided in Section 3.2 of this report.

In addition to these anticipated test modifications, changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this processor are described in Section 2.2.

For the conformity assessment of each Ada processor, a customized test suite is produced by the ACAL. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see Section 2.1), and possibly removing some inapplicable tests (see Section 2.1 and [UG97]).

1.3 DEFINITION OF TERMS

Acceptable result	A result that is explicitly allowed by the grading criteria of the test program for a grade of passed or inapplicable.
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	The means of checking conformity of Ada processors, consisting of tests, support programs, and a User's Guide. Also referred to as the Ada Conformity Assessment Test Suite.
Ada Conformity Assessment Test Suite (ACATS)	Alternate name for the ACVC (which see).
Ada Conformity Assessment Laboratory	An organization which carries out the procedures required to assess the conformity of an Ada processor.
Ada Conformity Assessment Authority (ACAA)	The organization that provides coordination and technical guidance for the Ada Conformity Assessment Laboratories.
Ada	An Ada processor.
Certified Status	(Also "certified as conforming") The status granted to an Ada processor by the award of an Ada Conformity Assessment Certificate.
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-wriiten 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.
Conformity Assessment	The process of checking the conformity of an Ada processor to the Ada programming language and of issing a certificate for that processor.
Customer	An individual or corporate entity who enters into an

	agreement with an ACAL which specifies the terms and conditions for ACAL services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or is attainable on the Ada processor for which certified status is realized.
Foundation Unit (Foundation Code)	An Ada package used by multiple tests. Foundation units are designed to be reusable. A valid foundation unit must be in the Ada library for those tests that are dependent on the foundation unit.
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 processor.
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.
Specialized Needs Annex	One of annexes C through H of [Ada95]. Testing of one or more specialized needs annexes is optional, and results for each tested annex are summarized in this report.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Unsupported Feature Test	A test for a language feature that is not required to be supported, because it is based upon a requirement stated in an Ada 95 Specialized Needs Annex.
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 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada processor. Reasons for a test's inapplicability may be supported by documents issued by the ISO known as Ada Commentaries and commonly referenced in the format AI95-ddddd. For this processor, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

C45322A, C45523A, and C45622A check that the proper exception is raised if MACHINE_OVERFLOWS is TRUE and the results of various floating-point operations lie outside the range of the base type; for this processor, MACHINE_OVERFLOWS is FALSE.

C4A012B checks that the proper exception is raised when FLOAT'MACHINE_OVERFLOWS is TRUE for negative powers of 0.0; for this processor, FLOAT'MACHINE_OVERFLOWS is FALSE.

C96005B uses values of type DURATION's base type that are outside the range of type DURATION; for this processor, the ranges are the same.

EA3004G checks whether Pragma Inline is obeyed for a function called from within a package specification. This processor does not obey Pragma Inline in this circumstance.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this processor does not support such sizes.

The tests listed in the following table check that USE_ERROR is raised if the given file operations are not supported for the given combination of mode and access method; this processor supports these operations.

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 an external sequential file is exceeded; this processor cannot restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of an external direct file is exceeded; this processor cannot restrict file capacity.

CE3115A checks operations on text files when multiple internal files are associated with the same external file and one or more are open for writing; USE_ERROR is raised when this association is attempted.

CE3304A checks that SET_LINE_LENGTH and SET_PAGE_LENGTH raise USE_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this processor.

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

CXB4009 depends on the availability of a COBOL compiler; the test environment did not include a Cobol compiler.

CXB5004..5 (2 tests) depend upon the availability of a Fortran compiler; the test environment did not include a Fortran compiler.

CXE2001, CXE4001..6 (6 tests), and LXE3001..2 (2 tests) check objectives related to inter-partition communication, requiring support for package System.RPC. This processor does not support package System.RPC, and the tests are rejected at compile time.

CXD2007, CXDB001..4 (4 tests), and LXD7008 check the functionality of Asynchronous Task Control. This processor does not support Asynchronous Task Control, and the tests are rejected at compile time.

2.2 MODIFICATIONS

In order to comply with the test objective it may be required to modify the test source code, the test processing method, or the test evaluation method. Modifications are allowable because at the time of test writing not all possible execution environments of the test and the capabilities of the compiler could be foreseen. Possible kinds of modification are:

- Test Modification: The source code of the test is changed.
  Examples for test modifications are the insertion of a pragma, the insertion of a representation clause, or the splitting of a B-test into several individual tests, if the compiler does not detect all intended errors in the original test.

- Processing Modification: The processing of the test by the Ada processor for
  conformity assessment is changed.
  Examples for processing modification are the change of the compilation order for a test that consists of multiple compilations or the additional compilation of a specific support unit in the library.

- Evaluation Modification: The evaluation of a test result is changed.
  An example for evaluation modification is the grading of a test other than the output from REPORT.RESULT indicates. This may be required if the test makes assumptions about implementation features that are not supported by the processor (e.g., the implementation of a file system on a bare target machine).

All modifications have been directed or approved by the ACAA after consulting the ACAL and the customer on the technical justification of the modification. All of the required test modifications from the "ACATS Modifications List", Version 2.1B were used along with any modifications detailed below.

Modifications were required for 58 tests.

The following 29 tests were split into two or more tests because this processor did not report the violations of the Ada Standard in the way expected by the original tests.

B32201A	B36201A	B393002	B41201A	B54A20A
B62001B	B67001A	B67001B	B67001C	B67001D
B74304A	B74304B	B74304C	B83E01C	B83E01D
B83E01E	B87B26A	B940003	B951001	B952001
BA11005	BA1101A	BA1101B	BA12008	BC3604A
BC3607A	BC51015	BC51016	BC51017	

The following 28 allowable test modifications from the "ACATS Modifications List", Version 2.1B were used.

C3A2A02	B490001	C760009	C760010	CD30002
CD30005	CXB3008	CXB4009	CXD1008	CXD2004
CXD6002	CXE5002	CXE5003	CXG2002	CXG2012
LXH4001	LXH4002	LXH4003	LXH4004	LXH4005
LXH4006	LXH4007	LXH4008	LXH4009	LXH4010
LXH4011	LXH4012			

CXG2014, as directed by the ACAA, was graded passed with the following code modification:

comment out line 345 (the call to Subtraction_Error_Test)

By effectively deleting this one line, the flawed subprogram will be removed from execution, and the other, valid checks can be made.

2.3 UNSUPPORTED FEATURES OF THE ADA 95 SPECIALIZED NEEDS ANNEXES

As allowed by [Ada95], a processor need not support any of the capabilities specified by a Specialized Needs Annex, or it may support some or all of them. For conformity assessment testing, each set of tests for a particular Annex is processed only upon customer request, but is processed in full (even if the Ada processor provides only partial support). As required by [Ada95], the failure to support a requirement of a Specialized Needs Annex must be indicated by a compile-time rejection or by raising a run-time exception. When a test for a Specialized Needs Annex thus indicates non-support, the result is graded "unsupported" (rather than "inapplicable"). However, if such a test is accepted and reports FAILED, the result is graded "failed", and is considered evidence of non-conformity.

All of the Specialized Needs Annexes were processed during this conformity assessment testing.

The following tests for Annex C, Systems Programming, were graded "unsupported": none.

The following tests for Annex D, Real-Time Systems, were graded "unsupported": none.

The following tests for Annex E, Distributed Systems, were graded "unsupported": none.

The following tests for Annex F, Information Systems, were graded "unsupported": none.

The following tests for Annex G, Numerics, were graded "unsupported": none.

The following tests for Annex H, Safety and Security, were graded "unsupported": none.

CHAPTER 3

PROCESSING INFORMATION

3.1 CONFORMITY ASSESSMENT PROCESS

A full evaluation of the customer's self-tested results was conducted at the ACAL's site.

Witness testing of this Ada processor was conducted at the customer-designated site by a representative of the ACAL.

A floppy diskette containing the customized test suite (see Section 1.3) was taken on-site by the ACAL representative for processing. The contents of the floppy diskette were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada processor.

The tests were compiled and linked on the host computer system, as appropriate. The executable images were transferred to the target computer system and run.

Testing was performed using command scripts provided by the customer and reviewed by the ACAL representative. See Appendix A for a complete listing of the processing options for this processor. Appendix A also indicates the default options.

The following explicit option settings were used during witness testing:

For B tests:

```
gcc -c -I$acvc_lib_dir
         -gnatE -gnato -gnatf -gnatvl -gnatq -gnatws -gnatd2 -gnatd7
```

For executable tests:

```
gcc -c -I$acvc_lib_dir -O0 -gnatE -gnato -gnatv -gnatws -gnatd7
gnatmake $main_name
       -I$acvc_lib_dir -O0 -gnatE -gnato -gnatv -gnatws -gnatd7
```

For L tests:

```
  gcc -c -I$acvc_lib_dir -O0 -gnatE -gnato -gnatv -gnatws -gnatd7
  gnatmake $main_name
         -I$acvc_lib_dir -O0 -gnatE -gnato -gnatv -gnatws -gnatd7
```

Test output, compiler and linker listings, and job logs were captured on floppy diskette and archived at the ACAL. The listings examined on-site by the ACAL representative were also archived.

3.2 MACRO PARAMETERS AND IMPLEMENTATION-SPECIFIC VALUES

This section contains the macro parameters used for customizing the ACATS. The meaning and purpose of these parameters are explained in [UG97]. 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 in a symbolic notation, using placeholders as appropriate.

3.2.1 Macro Parameters

Macro Parameter	Macro Value
$MAX_IN_LEN	200
$BIG_ID1	AAA ... A1 (200 characters)
$BIG_ID2	AAA ... A2 (200 characters)
$BIG_ID3	AAA ... A3A ... A (200 characters)
$BIG_ID4	AAA ... A4A ... A (200 characters)
$BIG_STRING1	"AAA ... A" (200/2 characters)
$BIG_STRING2	"AAA ... A1" ((200/2)-1 characters)
$BLANKS	" ... " (200-20 blanks)
$MAX_STRING_LITERAL	"AAA ... A" (200 characters)
$ACC_SIZE	32
$ALIGNMENT	2
$COUNT_LAST	2147483647

$ENTRY_ADDRESS	ENTRY_ADDR
$ENTRY_ADDRESS1	ENTRY_ADDR1
$ENTRY_ADDRESS2	ENTRY_ADDR2
$FIELD_LAST	255
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	86_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1	/NODIRECTORY/FILENAME
$ILLEGAL_EXTERNAL_FILE_NAME2	/@@/@@/@@\@@\@@\@@
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$LESS_THAN_DURATION	-86_400.0
$MACHINE_CODE_STATEMENT	Asm_Insn'(Asm ("nop"));
$MAX_INT	9223372036854775807
$MIN_INT	-9223372036854775808
$NAME	LONG_LONG_INTEGER
$NAME_SPECIFICATION1	/home//X2120A
$NAME_SPECIFICATION2	/home//X2120B
$NAME_SPECIFICATION3	/home//X3119A
$OPTIONAL_DISC	NO_SUCH_MACHINE_CODE_DISC
$RECORD_DEFINITION	RECORD ASM : STRING (1..4); END RECORD;
$RECORD_NAME	Asm_Insn
$TASK_SIZE	32
$TASK_STORAGE_SIZE	1024
$VARIABLE_ADDRESS	VAR_ADDR

PROCESSING INFORMATION

$VARIABLE_ADDRESS1	VAR_ADDR1
$VARIABLE_ADDRESS2	VAR_ADDR2

Package ImpDef and Its Children

The package ImpDef is used by several tests of core language features. Before use in testing, this package is modified to specify certain implementation-defined features. In addition, package ImpDef has a child package for each Specialized Needs Annex, each of which may need similar modifications. The child packages are independent of one another, and are used only by tests for their respective annexes.

This section presents the package ImpDef and each of the relevant child packages as they were modified for this conformity assessment. In the interests of simplifying this ACATR, the header comment block was removed from each of the package files.

3.2.1.1 Package ImpDef

```
-- IMPDEF.A
--
--!

with Report;
with Ada.Text_IO;
with System.Storage_Elements;

package ImpDef is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following boolean constants indicate whether this validation will
   -- include any of annexes C-H. The values of these booleans affect the
   -- behavior of the test result reporting software.
   --
   --    True  means the associated annex IS included in the validation.
   --    False means the associated annex is NOT included.

   Validating_Annex_C : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_D : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_E : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_F : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_G : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_H : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED
```

```
--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This is the minimum time required to allow another task to get
   -- control.  It is expected that the task is on the Ready queue.
   -- A duration of 0.0 would normally be sufficient but some number
   -- greater than that is expected.

   Minimum_Task_Switch : constant Duration := 0.1;
   --                                         ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This is the time required to activate another task and allow it
   -- to run to its first accept statement.  We are considering a simple task
   -- with very few Ada statements before the accept.  An implementation is
   -- free to specify a delay of several seconds, or even minutes if need be.
   -- The main effect of specifying a longer delay than necessary will be an
   -- extension of the time needed to run the associated tests.

   Switch_To_New_Task : constant Duration := 1.0;
   --                                        ^^^ -- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This is the time which will clear the queues of other tasks
   -- waiting to run.  It is expected that this will be about five
   -- times greater than Switch_To_New_Task.

   Clear_Ready_Queue : constant Duration := 5.0;
   --                                       ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Some implementations will boot with the time set to 1901/1/1/0.0
   -- When a delay of Delay_For_Time_Past is given, the implementation
   -- guarantees that a subsequent call to Ada.Calendar.Time_Of(1901,1,1)
   -- will yield a time that has already passed (for example, when used in
   -- a delay_until statement).

   Delay_For_Time_Past : constant Duration := 0.1;
   --                                         ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Minimum time interval between calls to the time dependent Reset
   -- procedures in Float_Random and Discrete_Random packages that is
   -- guaranteed to initiate different sequences.  See RM A.5.2(45).

   Time_Dependent_Reset : constant Duration := 0.3;
   --                                          ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Test CXA5013 will loop, trying to generate the required sequence
```

```
   -- of random numbers.  If the RNG is faulty, the required sequence
   -- will never be generated.  Delay_Per_Random_Test is a time-out value
   -- which allows the test to run for a period of time after which the
   -- test is failed if the required sequence has not been produced.
   -- This value should be the time allowed for the test to run before it
   -- times out.  It should be long enough to allow multiple (independent)
   -- runs of the testing code, each generating up to 1000 random
   -- numbers.

   Delay_Per_Random_Test : constant Duration := 1.0;
   --                                          ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The time required to execute this procedure must be greater than the
   -- time slice unit on implementations which use time slicing.  For
   -- implementations which do not use time slicing the body can be null.

   procedure Exceed_Time_Slice;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This constant must not depict a random number generator state value.
   -- Using this string in a call to function Value from either the
   -- Discrete_Random or Float_Random packages will result in
   -- Constraint_Error (expected result in test CXA5012).

   Non_State_String : constant String := "By No Means A State";
   --           MODIFY HERE AS NEEDED --- ^^^^^^^^^^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This string constant must be a legal external tag value as used by
   -- CD10001 for the type Some_Tagged_Type in the representation
   -- specification for the value of 'External_Tag.

   External_Tag_Value : constant String := "implementation_defined";
   --              MODIFY HERE AS NEEDED --- ^^^^^^^^^^^^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following address constant must be a valid address to locate
   -- the C program CD30005_1.  It is shown here as a named number;
   -- the implementation may choose to type the constant as appropriate.


   function Cd30005_Proc (X : Integer) return Integer;
   pragma Import (C, Cd30005_Proc, "_cd30005_1");

   pragma Linker_Options ("../cd300051.o");

   CD30005_1_Foreign_Address : constant System.Address:= Cd30005_Proc'Address;

   -- CD30005_1_Foreign_Address : constant System.Address:=
```

```
   --          System.Storage_Elements.To_Address ( 16#0000_0000# )
   --               --MODIFY HERE AS REQUIRED --- ^^^^^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constant must be the external name resulting
   -- from the C compilation of CD30005_1.  The string will be used as an
   -- argument to pragma Import.

   CD30005_1_External_Name : constant String := "_cd30005_1";
   --                   MODIFY HERE AS NEEDED --- ^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following constants should represent the largest default alignment
   -- value and the largest alignment value supported by the linker.
   -- See RM 13.3(35).

   Max_Default_Alignment : constant := Standard'Maximum_Alignment;
   --                                  ^ --- MODIFY HERE AS NEEDED

   Max_Linker_Alignment  : constant := Standard'Maximum_Alignment;
   --                                  ^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constants must be the external names resulting
   -- from the C compilation of CXB30130.C and CXB30131.C.  The strings
   -- will be used as arguments to pragma Import.

   CXB30130_External_Name : constant String := "CXB30130";
   --                   MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB30131_External_Name : constant String := "CXB30131";
   --                   MODIFY HERE AS NEEDED --- ^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constants must be the external names resulting
   -- from the COBOL compilation of CXB40090.CBL, CXB40091.CBL, and
   -- CXB40092.CBL.  The strings will be used as arguments to pragma Import.

   CXB40090_External_Name : constant String := "CXB40090";
   --                   MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB40091_External_Name : constant String := "CXB40091";
   --                   MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB40092_External_Name : constant String := "CXB40092";
   --                   MODIFY HERE AS NEEDED --- ^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constants must be the external names resulting
```

```
   -- from the Fortran compilation of CXB50040.FTN, CXB50041.FTN,
   -- CXB50050.FTN, and CXB50051.FTN.
   --
   -- The strings will be used as arguments to pragma Import.
   --
   -- Note that the use of these four string constants will be split between
   -- two tests, CXB5004 and CXB5005.

   CXB50040_External_Name : constant String := "args";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50041_External_Name : constant String := "tax";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50050_External_Name : constant String := "align";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50051_External_Name : constant String := "modify";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following constants have been defined for use with the
   -- representation clause in FXACA00 of type Sales_Record_Type.
   --
   -- Char_Bits should be an integer at least as large as the number
   -- of bits needed to hold a character in an array.
   -- A value of 6 * Char_Bits will be used in a representation clause
   -- to reserve space for a six character string.
   --
   -- Next_Storage_Slot should indicate the next storage unit in the record
   -- representation clause that does not overlap the storage designated for
   -- the six character string.

   Char_Bits         : constant := 8;
   --     MODIFY HERE AS NEEDED ---^

   Next_Storage_Slot : constant := 6;
   --     MODIFY HERE AS NEEDED ---^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constant must be the path name for the .AW
   -- files that will be processed by the Wide Character processor to
   -- create the C250001 and C250002 tests.  The Wide Character processor
   -- will expect to find the files to process at this location.

   Test_Path_Root : constant String :=
     "/data/ftp/public/AdaIC/testing/acvc/95acvc/";
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ --- MODIFY HERE AS NEEDED

   -- The following two strings must not be modified unless the .AW file
   -- names have been changed.  The Wide Character processor will use
   -- these strings to find the .AW files used in creating the C250001
```

```
   -- and C250002 tests.

  Wide_Character_Test : constant String := Test_Path_Root & "c250001";
  Upper_Latin_Test    : constant String := Test_Path_Root & "c250002";

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following instance of Integer_IO or Modular_IO must be supplied
   -- in order for test CD72A02 to compile correctly.
   -- Depending on the choice of base type used for the type
   -- System.Storage_Elements.Integer_Address; one of the two instances will
   -- be correct.  Comment out the incorrect instance.

   --M package Address_Value_IO is
   --M   new Ada.Text_IO.Integer_IO(System.Storage_Elements.Integer_Address);

   package Address_Value_IO is
         new Ada.Text_IO.Modular_IO(System.Storage_Elements.Integer_Address);

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef;


     --=========================================================================--


package body ImpDef is

   -- NOTE: These are example bodies.  It is expected that implementors
   --       will write their own versions of these routines.

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The time required to execute this procedure must be greater than the
   -- time slice unit on implementations which use time slicing.  For
   -- implementations which do not use time slicing the body can be null.

   Procedure Exceed_Time_Slice is
      T : Integer := 0;
      Loop_Max : constant Integer := 4_000;
   begin
      for I in 1..Loop_Max loop
         T := Report.Ident_Int (1) * Report.Ident_Int (2);
      end loop;
   end Exceed_Time_Slice;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef;
```

3.2.1.2 Package ImpDef.Annex_C

```
-- Version of IMPDEFC.A modified for ACT interrupt support
--
-- IMPDEFC.A
--
--!

with Ada.Interrupts.Names;

package ImpDef.Annex_C is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Interrupt_To_Generate should identify a non-reserved interrupt
   -- that can be predictably generated within a reasonable time interval
   -- (as specified by the constant Wait_For_Interrupt) during testing.

   Interrupt_To_Generate: constant Ada.Interrupts.Interrupt_ID :=
      Ada.Interrupts.Names.SIGPIPE;  -- to allow trivial compilation
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Wait_For_Interrupt should specify the reasonable time interval during
   -- which the interrupt identified by Interrupt_To_Generate can be
   -- expected to be generated.

   Wait_For_Interrupt : constant := 0.1;
   --                               ^^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Enable_Interrupts should enable interrupts, if this
   -- is required by the implementation. [See additional notes on this
   -- procedure in the package body.]

   procedure Enable_Interrupts;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Generate_Interrupt should generate the interrupt
   -- identified by Interrupt_To_Generate within the time interval
   -- specified by Wait_For_Interrupt. [See additional notes on this
   -- procedure in the package body.]

   procedure Generate_Interrupt;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_C;
```

```
     --=====================================================================--


package body ImpDef.Annex_C is

   -- NOTE: These are example bodies.  It is expected that implementors
   --       will write their own versions of these routines.

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Enable_Interrupts should enable interrupts, if this
   -- is required by the implementation.
   --
   -- The default body is null, since it is expected that most implementations
   -- will not need to perform this step.
   --
   -- Note that Enable_Interrupts will be called only once per test.

   procedure Enable_Interrupts is
   begin
      null;

   -- ^^^^^^^^^^^^^^^^^^^  MODIFY THIS BODY AS NEEDED  ^^^^^^^^^^^^^^^^^^^

   end Enable_Interrupts;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Generate_Interrupt should generate the interrupt
   -- identified by Interrupt_To_Generate within the time interval
   -- specified by Wait_For_Interrupt.
   --
   -- The default body assumes that an interrupt will be generated by some
   -- physical act during testing. While this approach is acceptable, the
   -- interrupt should ideally be generated by appropriate code in the
   -- procedure body.
   --
   -- Note that Generate_Interrupt may be called multiple times by a single
   -- test. The code used to implement this procedure should account for this
   -- possibility.

   procedure Generate_Interrupt is

      procedure c_kill (pid : Integer; sig : in Ada.Interrupts.Interrupt_ID);
      pragma Import (C, c_kill, "kill");

      function c_getpid return Integer;
      pragma Import (C, c_getpid, "getpid");

   begin
      Report.Comment (". >>>>> GENERATE THE INTERRUPT NOW <<<<< ");
      c_kill (c_getpid, Interrupt_To_Generate);
```

```
   -- ^^^^^^^^^^^^^^^^^^^  MODIFY THIS BODY AS NEEDED  ^^^^^^^^^^^^^^^^^^^
   end Generate_Interrupt;
--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--
end ImpDef.Annex_C;
```

PROCESSING INFORMATION

3.2.1.3 Package ImpDef.Annex_D

```
-- IMPDEFD.A
--
--!

package ImpDef.Annex_D is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This constant is the maximum storage size that can be specified
   -- for a task.  A single task that has this size must be able to
   -- run.  Ideally, this value is large enough that two tasks of this
   -- size cannot run at the same time.  If the value is too small then
   -- test CXDC001 may take longer to run.  See the test for further
   -- information.

   Maximum_Task_Storage_Size : constant := 16_000_000;
   --                                      ^^^^^^^^^^ --- MODIFY HERE AS
NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Indicates the type of processor on which the tests are running.
   -- Time_Slice indicates a uniprocessor with an operating system that
   -- simulates a multi-processor by using time slicing.

   type Processor_Type is (Uni_Processor, Time_Slice, Multi_Processor);

   Processor : constant Processor_Type := Uni_Processor;
   --                                     ^^^^^^^^^^^ --- MODIFY HERE AS
NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_D;
```

3.2.1.4 Package ImpDef.Annex_E

```
-- IMPDEFE.A
--
--!

package ImpDef.Annex_E is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The Max_RPC_Call_Time value is the longest time a test needs to wait for
   -- an RPC to complete.  Included in this time is the time for the called
   -- procedure to make a task entry call where the task is ready to accept
   -- the call.

   Max_RPC_Call_Time : constant Duration := 2.0;
   --                                       ^^^  --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_E;
```

3.2.1.5 Package ImpDef.Annex_G

```
-- IMPDEFG.A
--
--!

package ImpDef.Annex_G is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This function must return a "negative zero" value for implementations
   -- for which Float'Signed_Zeros is True.

   function Negative_Zero return Float;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_G;


     --=========================================================================--


package body ImpDef.Annex_G is


--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   --  This function must return a negative zero value for implementations
   --  for which Float'Signed_Zeros is True.
   --  We generate the smallest normalized negative number, and divide by a
   --  few powers of two to obtain a number whose absolute value equals zero
   --  but whose sign is negative.

   function Negative_Zero return Float is
      negz : float := -1.0 *
         float (float'Machine_Radix)
            ** ( Float'Machine_Emin - Float'Machine_Mantissa);
   begin
      return negz / 8.0;
   end Negative_Zero;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_G;
```

3.2.1.6 Package ImpDef.Annex_H

```
-- IMPDEFH.A
--
--!

package Impdef.Annex_H is

  type Scalar_To_Normalize is
       (   Id0,  Id1,  Id2,  Id3,  Id4,  Id5,  Id6,  Id7,  Id8,  Id9,
          Id10, Id11, Id12, Id13, Id14, Id15, Id16, Id17, Id18, Id19,
          Id20, Id21, Id22, Id23, Id24, Id25, Id26, Id27, Id28, Id29,
          Id30, Id31, Id32, Id33, Id34, Id35, Id36, Id37, Id38, Id39,
          Id40, Id41, Id42, Id43, Id44, Id45, Id46, Id47, Id48, Id49,
          Id50, Id51, Id52, Id53, Id54, Id55, Id56, Id57, Id58, Id59,
          Id60, Id61, Id62, Id63, Id64, Id65, Id66, Id67, Id68, Id69,
          Id70, Id71, Id72, Id73, Id74, Id75, Id76, Id77, Id78, Id79,
          Id80, Id81, Id82, Id83, Id84, Id85, Id86, Id87, Id88, Id89,
          Id90, Id91, Id92, Id93, Id94, Id95, Id96, Id97, Id98, Id99,
          IdA0, IdA1, IdA2, IdA3, IdA4, IdA5, IdA6, IdA7, IdA8, IdA9,
          IdB0, IdB1, IdB2, IdB3, IdB4, IdB5, IdB6 );

  -- NO MODIFICATION NEEDED TO TYPE SCALAR_TO_NORMALIZE.  DO NOT MODIFY.

  type Small_Number is range 1..100;

  -- NO MODIFICATION NEEDED TO TYPE SMALL_NUMBER.  DO NOT MODIFY.

--=====================================================================
  -- When the value documented in H.1(5) as the predictable initial value
  -- for an uninitialized object of the type Scalar_To_Normalize
  -- (an enumeration type containing 127 identifiers) is to be in the range
  -- Id0..IdB6, set the following constant to True; otherwise leave it set
  -- to False.

  Default_For_Scalar_To_Normalize_Is_In_Range : constant Boolean := False;
  --                                       MODIFY HERE AS NEEDED --- ^^^^^

--=====================================================================
  -- If the above constant Default_For_Scalar_To_Normalize_Is_In_Range is
  -- set True, the following constant must be set to the value documented
  -- in H.1(5) as the predictable initial value for the type
  -- Scalar_To_Normalize.

  Default_For_Scalar_To_Normalize : constant Scalar_To_Normalize := Id0;
  --                                       MODIFY HERE AS NEEDED --- ^^^

--=====================================================================
  -- When the value documented in H.1(5) as the predictable initial value
  -- for an uninitialized object of the type Small_Number
  -- (an integer type containing 100 values) is to be in the range
  -- 1..100, set the following constant to True; otherwise leave it set
  -- to False.
```

```
  Default_For_Small_Number_Is_In_Range : constant Boolean := False;
  --                                MODIFY HERE AS NEEDED --- ^^^^^

--=========================================================================
  -- If the above constant Default_For_Small_Number_Is_In_Range is
  -- set True, the following constant must be set to the value documented
  -- in H.1(5) as the predictable initial value for the type Small_Number.

  Default_For_Small_Number : constant Small_Number := 100;
  --                          MODIFY HERE AS NEEDED --- ^^^

--=========================================================================

end Impdef.Annex_H;
```

3.3 WITHDRAWN TESTS

At the time of this conformity assessment testing, the following 24 tests were withdrawn from the ACATS.

B37312B	BXC6A03	C390010	C392010	C392012	C42006A
C48009A	C760007	C760012	C761006	C761008	C761009
C9A005A	C9A008A	CD20001	CXC3004	CXD2005	CXD4009
CXD5002	CXDB005	CXDC001	CXG2022	E28002B	LA1001F

APPENDIX A

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

A.1 Compilation System Options

The compiler options of this Ada processor, 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.

Compilation System Options

Usage: gcc switches <sfile>

-gnata	Assertions enabled. Pragma Assert and pragma Debug to be activated
-gnatb	Generate brief messages to stderr even if verbose mode set
-gnatc	Check syntax and semantics only (no code generation attempted)
-gnate	Error messages generated immediately, not saved up till end
-gnatE	Generate full dynamic elaboration checks
-gnatf	Full errors. Multiple errors/line, all undefined references
-gnatg	GNAT style checks enabled
-gnati?	Identifier char set (?=1/2/3/4/8/p/f/n/w)
-gnatj?	Wide character encoding method (?=n/h/u/s/e)
-gnatknnn	Limit file names to nnn characters (k = krunch)
-gnatl	Output full source listing with embedded error messages
-gnatmnnn	Limit number of detected errors to nnn (1-999)
-gnatn	Inlining of subprograms (apply pragma Inline across units)
-gnatN	Inline all subprogram calls
-gnato	Enable optional checks (overflow, stack check, elaboration checks
-gnatp	Suppress all checks
-gnatP	Enable generation of polling
-gnatq	Don't quit, try semantics, even if parse errors
-gnatr	Reference manual column layout required
-gnats	Syntax check only
-gnatt	Tree output file to be generated
-gnatu	List units for this compilation
-gnatv	Verbose mode. Full error output with source lines to stdout
-gnatw?	Warning mode. (?=s/e for suppress/treat as error)
-gnatW	Set wide character encoding method

```
  -gnatx?    Cross-reference level and switches (?=1/2/3/4/5/9/b/s)
  -gnatz?    Distribution stub generation (r/s for receiver/sender stubs)
  -gnat83    Enforce Ada 83 restrictions

Debug flags for compiler:
-------------------------

  -gnatda   Generate messages tracking semantic analyzer progress
  -gnatdb   Show encoding of type names for debug output
  -gnatdc   List names of units as they are compiled
  -gnatdd   Dynamic allocation of tables messages generated
  -gnatde   List the entity table
  -gnatdf   Full tree/source print (includes withed units)
  -gnatdg   Print source from tree (generated code only)
  -gnatdh   Generate listing showing loading of name table hash chains
  -gnatdi   Generate messages for visibility linking/delinking
  -gnatdj   Suppress "junk null check" for access parameter values
  -gnatdk   Generate GNATBUG message on abort, even if previous errors
  -gnatdl   Generate unit load trace messages
  -gnatdm   Allow VMS features even if not OpenVMS version
  -gnatdn   Generate messages for node/list allocation
  -gnatdo   Print source from tree (original code only)
  -gnatdp   Generate messages for parser scope stack push/pops
  -gnatdr   Generate parser resynchronization messages
  -gnatds   Print source from tree (including original and generated stuff)
  -gnatdt   Print full tree
  -gnatdu   Uncheck categorization pragmas
  -gnatdv   Output trace of overload resolution
  -gnatdw   Print trace of semantic scope stack
  -gnatdx   Force expansion on, even if no code being generated
  -gnatdy   Print tree of package Standard
  -gnatdz   Print source of package Standard
  -gnatdB   Output debug encoding of type names and variants
  -gnatdE   Apply elaboration checks to predefined units
  -gnatdG   Do not compile generics
  -gnatdL   Output trace information on elaboration checking
  -gnatdP   Do not check for controlled objects in preelaborable packages
  -gnatdX   Force use of zero-cost exception approach
  -gnatd1   Error msgs have node numbers where possible
  -gnatd2   Eliminate error flags in verbose form error messages
  -gnatd3   Dump bad node in Comperr on an abort
  -gnatd4   Inhibit automatic krunch of predefined library unit files
  -gnatd5   Debug output for tree read/write
  -gnatd6   Default access unconstrained to thin pointers
  -gnatd7   Do not output version & file time stamp in -gnatv or -gnatl mode
  -gnatd8   Force opposite endianness in packed stuff


General GCC options applicable to GNAT:
---------------------------------------

  -c       Compile or assemble the source files, but do not link.
  -O0      No code optimization (this is the default setting)
  -O1      Optimize.
```

```
  -O2       Optimize even more.
  -O3       Optimize yet more.

  -S        Stop after the stage of compilation proper; do not assemble.
            The output is an assembler code file for each non-assembler
            input file specified.

  -mieee    Generate ieee754 compatible code.

  -o file   Place output in file file. This applies regardless to whatever
            sort of output GCC is producing, whether it be an executable
            file or an object file.

  -v        Print the commands executed to run the stages of compilation.
            Also print the version number of the compiler driver program
            and of the preprocessor and the compiler proper.

  -g        Produce debugging information in the operating system's native
            format. GDB can work with this debugging information.

  -Bprefix  This option specifies where to find the executables, libraries
            and data files of the compiler itself.

  -Idir     Specify library and source files search path

  -Ldir     Add directory dir to the list of directories to be searched
            for `-l'.
```

A.2 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.

```
Linker Options
--------------

Usage: gnatbind switches lfile

  -aOdir  Specify library files search path
  -aIdir  Specify source files search path
  -A      Generate binder program in Ada
  -b      Generate brief messages to stderr even if verbose mode set
  -c      Check only, no generation of binder output file
  -C      Generate binder program in C (default)
  -e      Output complete list of elaboration order dependencies
  -f      Full elaboration semantics. Follow Ada rules. No attempt to be kind
  -h      Horrible (worst-case) elaboration order
  -Idir   Specify library and source files search path
  -I-     Don't look for sources & library files in default directory
```

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

```
  -l      Output chosen elaboration order
  -mnnn   Limit number of detected errors to nnn (1-999)
  -n      No main program
  -o file give the output file name (default is b_xxx.c)
  -r      Rename generated main program from main to gnat_main
  -s      Require all source files to be present
  -t      Tolerate time stamp and other consistency errors
  -Tn     Enable or Disable time-slicing
  -v      Verbose mode. Error messages,header, summary output to stdout
  -wx     Warning mode. (x=s/e for suppress/treat as error)
  -x      Exclude source files (check object consistency only)
  lfile   Library file names


Usage: gnatmake  opts  name  {[-cargs opts] [-bargs opts] [-largs opts]}

  name  is a file name from which you can omit the .adb or .ads suffix

gnatmake switches:
  -a       Consider all files, even readonly ali files
  -c       Compile only, do not bind and link
  -f       Force recompilations of non predefined units
  -i       In place. Replace existing ali file, or put it with source
  -jnum    Use nnn processes to compile
  -k       Keep going after compilation errors
  -m       Minimal recompilation
  -M       List object file dependences for Makefile
  -n       Check objects up to date, output next file to compile if not
  -o name  Choose an alternate executable name
  -q       Be quiet/terse
  -v       Motivate all (re)compilations

  --GCC=command       Use this gcc command
  --GNATBIND=command  Use this gnatbind command
  --GNATLINK=command  Use this gnatlink command

Gnat/Gcc switches such as -g, -O, -gnato, etc.are directly passed to gcc

Source & Library search path switches:
  -aLdir  Skip missing library sources if ali in dir
  -Adir   like -aLdir -aIdir
  -aOdir  Specify library/object files search path
  -aIdir  Specify source files search path
  -Idir   Like -aIdir -aOdir
  -I-     Don't look for sources & library files in the default directory
  -Ldir   Look for program libraries also in dir

To pass an arbitrary switch to the Compiler, Binder or Linker:
  -cargs opts   opts are passed to the compiler
  -bargs opts   opts are passed to the binder
  -largs opts   opts are passed to the linker
```

APPENDIX B

POINTS OF CONTACT

Ada Conformace Assessment Laboratory

Phil Brashear
EDS Conformance Testing Center
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton OH 45424-0593
U.S.A.
Phone : (937) 237-4510
Internet : brashp@dysmailpo.deisoh.msd.eds.com

AdaConformity Assessment Authority

Randall Brukardt
ACAA
P.O. Box 1512
Madison WI 53701
U.S.A.
Phone : (608) 245-0375
Internet : Rbrukardt@bix.com

For technical information about this Ada processor, contact:

Robert Dewar
Ada Core Technologies, Inc.
73 Fifth Ave., Suite 11B
New York NY 10003
(212) 620-7300 (ext 100)
dewar@gnat.com

POINTS OF CONTACT

For sales information about this Ada processor, contact:

Karen Syck
Ada Core Technologies, Inc.
73 Fifth Ave., Suite 11B
New York NY 10003
(212) 620-7300 (ext 118)
syck@gnat.com or sales@gnat.com

APPENDIX C

REFERENCES

[ACAP] Ada Conformity Assessment Procedures, Version 1.1, EDS Conformance Testing Center, September 1998

[Ada95] Reference Manual for the Ada Programming Language, ANSI/ISO/IEC 8652:1995

[Pro98] Ada Conformity Assessment Authority Operating Procedures, Version 1.3, Ada Resource Association, October 1998

[UG97] The Ada Compiler Validation Capability Version 2.1 User's Guide, Revision 1, SAIC and CTA, March 1997

REFERENCES

end of document

(REMOVE THIS PAGE)