ACAL Control Number: EDS19981207INT01-2.1

Ada Conformity Assessment Test Report

Certificate Number: A990209E2.1-048
AverStar Inc.
AdaMagic with C Intermediate version 3.92
Sun SPARCstation 10 under Solaris 2.5 with gcc 2.7.2

(Final)
11 February 1999

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.

TABLE OF CONTENTS

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 A990209E2.1-048 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_1F) (See Section 2.2 for details)
Location of Testing	AverStar Inc. 23 Fourth Avenue Burlington MA 01803
Test Completion Date	9 February 1999

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 A990209E2.1-048

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	24	2	21	1	0
& required Section 13	161	1	160	0	0
(representation support)	---	---	---	---	---
	185	3	181	1	0
D Real-Time Systems (which requires Annex C)	58	5	** not tested **		
E Distributed Systems	26	0	** not tested **		
F Information Systems	21	0	** not tested **		
G Numerics	29	1	** not tested **		
H Safety and Security	30	0	** not tested **		

DECLARATION OF CONFORMITY

Customer:	AverStar 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:	AdaMagic with C Intermediate version 3.92
Host Computer System:	Sun SPARCstation 10 Solaris 2.5 with gcc 2.7.2
Target Computer System:	Same as host

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

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 Conformity	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.

The following 17 tests check for the predefined type SHORT_INTEGER; for this processor, there is no such type:

B36105C	C45231B	C45304B	C45411B	C45502B
C45503B	C45504B	C45504E	C45611B	C45613B
C45614B	C45631B	C45632B	B52004E	C55B07B
B55B09D	CD7101E			

C45231D and CD7101G check for a predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER; for this processor, there is no such type.

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.

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

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.

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

BD8001A, BD8002A, BD8003A, BD8004A..C (3 tests), and AD8011A use machine code insertions; this processor provides no package MACHINE_CODE.

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
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_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
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.

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.

CXB4001..9 (9 tests) depend on the availability of an interface to COBOL; this processor does not support Cobol interfaces. (See Section 2.2 re CXB4001, CXB4007, and CXB4009.)

CXB5004..5 (2 tests) depend upon the existence of convention Fortran; this processor rejects the use of convention Fortran in Pragma Import.

CXC6001 checks for incorrect usages of atomic and volatile elementary types. This processor does not support indivisible read/update for some types; the application of pragma atomic to a record type in line 65 is rejected at compile time by this processor.

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:

o 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.

o 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.

o 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 2.1F were used along with any modifications detailed below.

Modifications were required for 20 tests.

The following 5 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.

B2A007A	B32201A	B44004C	BA1101E	BC2001D

CA5004B was split into multiple files because when any unit of a compilation (file) is made obsolete or is replaced, this processor removes all units in that file from the environment, as permitted by [Ada95] 10.1(4). File CA5004B0 was split, at line 61, into two separate files.

The following 12 allowable test modifications from the "ACATS Modifications List", Version 2.1F were used.

B830001	C330001	C332001	C460008	CD10001
CD2A53A	CD30002	CD30003	CD92001	CDE0001
CXAA016	CXC7001			

CD33002, as directed by the ACAA, was graded passed by code & processing modifications. This test checks that various Component_Sizes are able to be specified, with the proper results. But the Component_Size value specified at line 74 exceeds what this implementation must support (cf. AI95-00109/07), and so is rejected at compile time. This test was also processed with lines 73 & 74 commented out; the modified test was passed. The modified test can be found with VCS label A2_1F_002 in the ACATS Version Control System.

CXB3013 assumes the existence of a C function "strdup" that is not an ANSI C standard function. The C compiler used in the testing does not provide such a function; hence file CXB30131.C was modified by inserting, at line 56, a definition of "strdup" that provides the expected functionality:

```
        char *strdup(char *s)
             char *result = (char *) malloc(sizeof(char)*strlen(s));
             return strcpy(result,s);
```

The modified test can be found with VCS label A2_1F_002 in the ACATS Version Control System.

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.

The set of tests for each of the following Specialized Needs Annexes was not processed during this conformity assessment testing:

Annex D, Real-Time Systems	(all BXD*, CXD*, & LXD* files)
Annex E, Distributed Systems	(all BXE* & CXE* files)
Annex F, Information Systems	(all BXF* & CXF* files)
Annex G, Numerics	(all CXG* files)
Annex H, Safety and Security	(all BXH*, CXH*, & LXH* files)

No tests for Annex C, Systems Programming, were graded "unsupported".

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, linked, and executed on the host computer system.

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 adareg

Option/Switch	Effect
-q	Quiet mode -- suppress all inessential messages.

For compile

Option/Switch	Effect
-lc	Continuous source listing interspersed with messages.
+mr msg_kind	Enables the display of any messages of msg_kind for any recursive invocations of the compiler.
-eo	Turns on emitter optimizations (default).
-q	Quiet mode -- suppress all inessential messages.

For build

Option/Switch	Effect
-q	Quiet mode -- suppress all inessential messages.

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	1
$COUNT_LAST	2147483647
$ENTRY_ADDRESS	FCNDECL.DATA(4)'ADDRESS
$ENTRY_ADDRESS1	FCNDECL.DATA(5)'ADDRESS
$ENTRY_ADDRESS2	FCNDECL.DATA(6)'ADDRESS
$FIELD_LAST	2147483647
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	86_401.0
$ILLEGAL_EXTERNAL_FILE_NAME1	/NODIRECTORY/FILENAME
$ILLEGAL_EXTERNAL_FILE_NAME2	/NODIRECTORY2/FILENAME2
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$LESS_THAN_DURATION	-90_000.0
$MACHINE_CODE_STATEMENT	NULL;
$MAX_INT	2147483647
$MIN_INT	-2147483648

PROCESSING INFORMATION

$NAME	NO_SUCH_TYPE_AVAILABLE
$NAME_SPECIFICATION1	X2120A
$NAME_SPECIFICATION2	X2120B
$NAME_SPECIFICATION3	X3119A
$OPTIONAL_DISC	OPTIONAL_DISC
$RECORD_DEFINITION	RECORD DUMMY : INTEGER; END RECORD;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	64
$TASK_STORAGE_SIZE	2048
$VARIABLE_ADDRESS	FCNDECL.DATA(1)'ADDRESS
$VARIABLE_ADDRESS1	FCNDECL.DATA(2)'ADDRESS
$VARIABLE_ADDRESS2	FCNDECL.DATA(3)'ADDRESS

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 := False;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

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

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

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

   Validating_Annex_H : constant Boolean := False;
   --                                       ^^^^^ --- 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 := 0.5;  -- ewcc target
   --                                        ^^^ -- 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.

   --CD30005_1_Foreign_Address : constant System.Address :=
System.Null_Address;
   --        MODIFY HERE IF NEEDED --- ^
   --          MODIFY HERE AS REQUIRED --- ^^^^^^^^^^^^^

   function CD30005_1_Foreign_Address return System.Address;
   pragma import (c, CD30005_1_Foreign_Address, "_cd30005_2", "_cd30005_2");

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

   -- 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 := 1;
   --                                  ^ --- MODIFY HERE AS NEEDED

   Max_Linker_Alignment  : constant := 1;
   --                                  ^ --- 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 := "CXB50040";
   --                 MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50041_External_Name : constant String := "CXB50041";
   --                 MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50050_External_Name : constant String := "CXB50050";
   --                 MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50051_External_Name : constant String := "CXB50051";
   --                 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
     "V:\ADA_MAGIC\ACVC_21\ADI\C2\";

   -- 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.

   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 := 100;
   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

```
-- 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.Interrupt_ID'First;  -- to allow trivial compilation
      Ada.Interrupts.Names.SIGINT;
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  --- 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 := 10.0;
   Wait_For_Interrupt : constant := 1.0;
   --                              ^^^^ --- 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;


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

```
with System.RTS.TGT.Kernel.Interrupts;
with Report;

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  ^^^^^^^^^^^^^^^^^^^^
      -- No modifications needed; interrupts are normally enabled.

   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
   begin
      Report.Comment (". >>>>> GENERATE THE INTERRUPT NOW <<<<< ");

      System.RTS.TGT.Kernel.Interrupts
        .Generate_Interrupt(Interrupt_To_Generate);

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

   end Generate_Interrupt;
```

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

end ImpDef.Annex_C;
```

3.3 WITHDRAWN TESTS

At the time of this conformity assessment testing, the following 25 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	EA3004G
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.

```
Usage: adareg [options ...] [file_or_directory ...]

    file_or_directory    A file or a directory name (string).  There
                         may be any number of these.  Any file is
                         registered.  For a directory, all recognized
                         Ada source files in that directory that have
                         not been registered, or have been modified
                         since last registration, are registered.

Options Summary:

    -help or -h         Display this help message.
    -v                  Run verbosely.
    -0                  Identifies executable version number.
    -q                  Quiet mode -- suppress all inessential messages..
    -cl                 Create the library file if it does not exist.
    -s unit_name        Autoregister files in search of 'unit_name' DECL.
    -b unit_name        Autoregister files in search of 'unit_name' BODY.
    -all                Explicitly register all new source files in
                        all source directories of the library.
    -no                 Do not invalidate old files when explicitly
                        registering.
    -ut unit_name unit_type
                        Autoregister files in search of 'unit_name' of
                        'unit_type'.  This is for internal use, when
                        adareg is called by other tools.
    unit_name           A string.
    unit_type           An enumeration integer corresponding to a unit type.

    Options and file_or_directories are cumulative and processed in
```

the order given, except that any explicit registration request causes all autoregister requests to be ignored. See user documentation for more information on explicit and implicit (auto) registration.

```
Usage: adacsrc [options ...] [file ...]

    file                A source file to be compiled.  There may be
                        multiple files specified.

Options Summary:

    -help or -h         Display this help message.

Listing Options

    -lc                 Continuous source listing interspersed with messages.
    -le                 Source listing only if there are errors.
    -lf filename        Use 'filename' for listing, instead of default.
    -lp                 Paginated source listing interspersed with messages.
    -lr                 Relevant-only source listing, (only source lines
                        for which there are error or warning messages).
    -lx                 Cross reference listing  (turns on -xr).
    -pl length          Set page length of source listing file to length.
    -pw width           Set page width of source listing file to width.

Message Options

    -m  msg_kind        Suppresses the display of any messages of msg_kind
                        for the current invocation of the compiler.
    +m  msg_kind        Enables the display of any messages of msg_kind
                        for the current invocation of the compiler.
    -mr msg_kind        Suppresses the display of any messages of msg_kind
                        for any recursive invocations of the compiler.
    +mr msg_kind        Enables the display of any messages of msg_kind
                        for any recursive invocations of the compiler.

    The valid values for msg_kind:
        a - all messages
        d - implementation-dependent warning messages
        e - error messages
        i - information messages
        n - nyi messages
        w - general warning messages
        r - redundant messages

    By default, all messages except information and redundant messages are
    displayed. For recursive invocations, no messages are displayed by
    default.  For convenience, "-m a" will suppress all messages *except*
    errors.

Miscellaneous Options

    -a                  Analyzer only, don't run the Emitter or BackEnd.
    +bw                 Enable BackEnd warnings.
    -c                  Frontend only, don't run the BackEnd.
    -e count            Stop reporting errors after the count but keep
                        on going.
    -eo                 Turns on emitter optimizations (default).
```

-f	Force generation of .c despite any errors.
-g	Generate information for symbolic debugger.
-ga	Generate FrontEnd debugging information.
-gc	Generate BackEnd debugging information.
-late_inlines	Allow pragma Inline to be specified after a specless subprogram body (for compatibility with Ada 83).
-N	Suppress certain numeric checks.
-noeo	Turns off emitter optimizations.
-noxr	Do no save xref info for the Browser.
-nonr	Force nr off.
-nr	No "current heap" releases (just keep allocating, never releasing).
-nz	Initialize all heap memory to a non-zero value. (In hex, the non-zero value is BAD1BAD1 so it is easy to spot in the debugger, and causes a Bus Error on the Sparc when dereferenced.)
-O level	Call the optimizer with optimizing level: all, 3, 2, 1, 0, none. The default is to use level 3 optimization.
-prl	Record layout listing for packed record types only.
-rl	Record layout listing for all record types.
-s	Suppress all checks.
-t	Trace each declaration and statement passed to the emitter.
-xr	Save xref info for the Browser (default).

Driver Options

-0	Identifies executable version numbers (default).
-ke	Keep intermediate files.
-ki	Keep the info file.
-ne	Don't re-exec adacomp process on failure.
-nl	Don't re-exec adacomp process on last file.
-of file	Read options from specified "file".
-pB "option"	Pass "option" to the Back End.
-q	Quiet mode -- suppress all inessential messages.
-sr	(Search and Register) Enable automatic registration of source files.
-T	Print timing information on compiler phases.
-v	Verbose mode -- driver reports its every action.

Executable File Overrides

-xd dir_path	Override default ADA_MAGIC environment variable if exists.
-xL exe_path	Override default lister.
-xO exe_path	Override default optimizer.
-xX exe_path	Override default xref lister.

(See manual for more details.)

A.2 Linker Options

The linker 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 linker documentation and not to this report.

```
Usage: adacbuild [options ...] [unit ...]

    unit           The main unit to be linked.  There may be
                   multiple units specified.

Options Summary:

    -help or -h    Display this help message.

    -0             Identifies executable version numbers (default).
    +bw            Enable linker warnings.
    -f             Force linking, despite any prelinker errors.
    -g             Build with debugging symbols.
    -ke            Keep intermediate files.
    -ll option     Pass "option" to linker.
    -na            No autoregistration.
    -nc            No recompilations.
    -nl            No link (prelink, but do not call linker).
    -no            No ".o out of date" recompilations.
    -o file        Place linked output in "file" instead of using the
                   default filename.
    -ol object     Pass "object" file to linker.
    -pre unit      Preelaborate "unit".
    -pru unit      Use certain pragmas of "unit" to override main unit
                   pragmas.
    -q             Quiet mode -- suppress all inessential messages.
    -r             Use a more "friendly" elaboration order.
    -v             Provide verbose output.
```

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

The following pages contain a sample test script.

```
==============================================================================
====

Testing Script

==============================================================================
====

#!/bin/csh -f


#quick help for script
if ("$1" == "" ) then
    echo "sample_acvc_test_script procedure_name [source_files...] \
 \
     procedure_name == name of ADA procedure to build \
     source_files  == ADA source files to compile \
\
     execution output of procedure is placed in a .out file \
"
    exit
endif

# parse args to scripts
#first arg
set PROCEDURE="$1"
shift
#rest of args
set SOURCE="$*"


#script variables
set ACVC_SUPPORT_DIR = /home/dsd/h/cpl2/ACVC_21/gcc/support

#set up tools to run
setenv ADA_MAGIC /home/dsd/f/cpl2/releases/ADA_OPT_MAGIC
set COMPILER = $ADA_MAGIC/bin/adacsrc
set BUILDER  = $ADA_MAGIC/bin/adacbuild
set REGISTER = $ADA_MAGIC/bin/adareg

#compile sources
if ( "$SOURCE" != "" ) then

    #register support files
    echo -----------
    echo Registering support files
    echo -----------
    $REGISTER $ACVC_SUPPORT_DIR

    #compile ADA source
    echo ----------
    echo Compiling: $SOURCE
    echo ----------
```

```
    $COMPILER $SOURCE

endif

#build ADA procedure
echo ---------
echo Building: $PROCEDURE
echo ---------
$BUILDER $PROCEDURE

#resulting executable from build
set EXECUTABLE = $PROCEDURE.exe
set EXE_OUTPUT = $PROCEDURE.out

#execute executable if build was successful
if (! -e $EXECUTABLE ) then
    echo ----------
    echo Executable not found!
    echo Build must have failed.
    echo ----------
    exit
endif

#execute procedure
echo ----------
echo Executing: $EXECUTABLE
echo ----------
$EXECUTABLE > $EXE_OUTPUT

#echo results to screen
cat $EXE_OUTPUT

#were done
echo -----
echo Done: $PROCEDURE
echo -----

#clean up, delete executable
rm $EXECUTABLE



#
# brief command explanation
#
# REGISTER Registers ADA source files into the current program library.
# COMPILER Compiles the ADA source files into objects.
# BUILDER  Links the required objects needed for the specified procedure.
#          If some required files have not been compiled, the builder
#          automaticly invokes the compiler on the required ADA source files.
# cat      Echos the contents of the given file to the screen.
# rm       Deletes the specified file.
#
#
```

APPENDIX B

POINTS OF CONTACT

Ada Conformance 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

Ada Conformity 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:

Bob Duff
AverStar Inc.
23 Fourth Avenue
Burlington MA 01803-3303
(781)-221-6990
bobduff@averstar.com

POINTS OF CONTACT

For sales information about this Ada processor, contact:

Steve Dubin
AverStar Inc.
23 Fourth Avenue
Burlington MA 01803-3303
(781)-221-6990
dubin@averstar.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)