AVF Control Number: EDS19970918GHS10-2.1
DATE COMPLETED
BEFORE ON-SITE: 27 MAR 98
AFTER ON-SITE: 29 MAY 98

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 980403e2.1-010
Green Hills Software, Inc.
Green Hills Optimizing Ada95 PowerPC Compiler, 1.8.8E
Sun SPARCstation 5 under Solaris, 2.5 =>
Motorola MVME 1603 (PowerPC 603) under VelOSity 1.0

(Final)

Prepared By:
Ada Validation Facility
Electronic Data Systems
4646 Needmore Road, Bin #46
P.O. Box 24593
Dayton, OH 45424-0593

TABLE OF CONTENTS

PREFACE

This report documents the validation testing of an Ada 95 implementation. This testing was conducted according to the Ada Compiler Validation Procedures version 5.0 using the Ada Compiler Validation Capability test suite version 2.1, and completed 3 April 1998.

The successful completion of validation testing is the basis for the Ada certification body's issuance of a validation certificate and for subsequent registration of derived implementations. A copy of the validation certificate 980403e2.1-010 and its attachment which were awarded for this validation are presented on the following two pages. Validation testing does not ensure that an implementation has no nonconformities to the Ada 95 standard other than those, if any, documented in this report. The compiler vendor declares that the tested implementation contains no deliberate deviation from the Ada 95 standard; a copy of this Declaration of Conformance is presented immediately after the certificate pages.

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.

Phil Brashear
Manager, Ada Validation Facility
EDS Conformance Testing Center
4646 Needmore Road, Bin #46
P.O. Box 24593
Dayton, OH 45424-0593
U.S.A.

Ada Validation Organization
Director, Computer and Software
Engineering Division
Institute for Defense Analyses
Alexandria VA 22311
U.S.A.

Ada Joint Program Office
Director
Center for Information Management
Defense Information Systems Agency
Alexandria VA 22041
U.S.A.

(Insert copy of certificate here)

Specialized Needs Annexes

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

SPECIALIZED NEEDS ANNEXES	Passed	Inappli-cable	Unsup-ported	With-Drawn	Total
C Systems Programming & required Section 13 (representation support)	21 159 --- 180	1 1 --- 2	0 0 --- 0	2 1 --- 3	24 161 --- 185
D Real-Time Systems	55	0	0	3	58
E Distributed Systems	0	0	26	0	26
F Information Systems	0	0	21	0	21
G Numerics	0	0	29	0	29
H Safety and Security	0	0	30	0	30

DECLARATION OF CONFORMANCE

Customer: Green Hills Software, Inc.

Ada Validation Facility: Electronic Data Systems
4646 Needmore Road, Bin #46
P.O. Box 24593
Dayton, OH 45424-0593
U.S.A.

ACVC Version: 2.1

Ada Implementation

Ada Compiler Name and Version: Green Hills Optimizing Ada95
PowerPC Compiler, 1.8.8E

Host Computer System: Sun SPARC Station 5
Solaris, 2.5

Target Computer System: Motorola MVME 1603 (PowerPC 603)
VelOSity 1.0

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 Validation Summary Report.

_____ _____

Customer Signature Date

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro97] against the Ada Standard [Ada95] using the Ada Compiler Validation Capability (ACVC) Version 2.1. This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro97]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG97].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). Validated status is awarded only to the implementation identified in this report. Copies of this report are available to the public from the AVF that performed this validation.

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to the Ada Validation Organization. For all points of contact see Appendix B.

1.2 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and 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, validation testing is discontinued.

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

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. 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 ACVC, 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 implementation, 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 implementation are described in Section 2.2.

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

1.3 LEGACY TESTS

ACVC 2.1 consists of legacy tests and tests specific to Ada 95. The legacy tests were taken from ACVC 1.12 with possibly minor modifications to remove incompatibilities with Ada 95. The remaining tests were developed in order to test new features of Ada 95. A consequence of this approach is that the naming conventions for tests are not uniform. The test name of a legacy test always refers to the Ada 83 Standard, even if the feature covered by the test was moved to a different section in [Ada95].

1.4 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 (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide, and the template for the Validation Summary Report.
ACVC Maintenance Organization (AMO)	The part of the certification body that maintains the ACVC.
Ada Implementation	An Ada compilation system, including any required runtime support software, together with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Certification Body	The organizations (AJPO, AVO, AVFs), collectively responsible for defining and implementing Ada validation policy, including production and maintenance of the ACVC tests, and awarding of Ada validation certificates.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic

	operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or is attainable on the Ada implementation for which validation 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 implementation.
ISO	International Organization for Standardization.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management.
Specialized Needs Annex	One of annexes C through H of [Ada95]. Validation against one or more specialized needs annexes is optional. For each annex, there is a test set that applies to it. In addition to all core language tests, the appropriate set of tests must be processed satisfactorily for an implementation to be validated for a specialized needs annex.
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.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro97].

Validation — The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.

Withdrawn Test — A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 INAPPLICABLE TESTS

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

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 implementation, 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 implementation, 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 implementation, FLOAT'MACHINE_OVERFLOWS is FALSE.

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

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

CD30002 checks for correct implementation of various alignments, some of which need not be supported. This implementation rejects the alignment clause at line 130. (See section 2.2.)

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 implementation 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 implementation cannot restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of an external direct file is exceeded; this implementation 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 implementation.

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

CXB4001..9 (9 tests) depend on the availability of an interface to COBOL; this implementation does not support Cobol interfaces.

CXB5001..5 (5 tests) depend upon the availability of an interface to Fortran; this implementation does not support Fortran interfaces.

CXC6001 checks for incorrect usages of atomic and volatile elementary types. This implementation 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 implementation.

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 implementation for validation 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 implementation (e.g., the implementation of a file system on a bare target machine).

All modifications have been directed by the AVO after consulting the AVF and the customer on the technical justification of the modification.

Modifications were required for 24 tests (BXC6A04 is listed twice).

The following 11 tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

B32201A	B44004C	B830001	B83E01C	B83E01D
B83E01E	BA1101E	BA21003	BA3006A	BC2001D
BXC6A04				

C761007, as directed by the AVO, was graded passed with the following Code Modification:

```
replace line 376
  TCTouch.Validate( "GHGHIJ", "Asynchronously aborted operation" );
with:
  TCTouch.Validate( "GHIJ", "Asynchronously aborted operation" );
```

The original code will cause the check at line 376 to be failed because the procedures C761007_0.Finalize (@87ff) and C761007_1.Finalize (@133ff) both ensure that no duplicate characters are put into the check string. (The AVO requires this change so to retain this test for finalization, as several related test programs are withdrawn.)

C980001, as directed by the AVO, was graded passed with the following Code Modification:

```
comment out lines 251 & 274 (=> -- C980001_0.Hold_Up.Lock )
```

This modification is necessary in order to prevent the test from hanging with a queued call to the protected object C980001_0.Hold_Up.

EA3004G was graded passed by Grading Modification as directed by the AVO. This test expects the reference to an obsolete unit to be detected at compile time; this implementation makes the detection at link time.

CD30002 was graded inapplicable by Grading Modification as directed by the AVO. This test checks that various Alignments are able to be specified, with the proper results. This implementation does not support the double-word alignment given at line 130 for an integer object, which it rejects at compile time. Whether an implementation that fully supports the Systems Programming Annex (C) must accept such a representation clause was not decided before validation time, and so this test was graded inapplicable (vs. "unsupported").

CD33002 was graded passed by Code Modification as directed by the AVO. 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.

CXA5015, as directed by the AVO, was graded passed with the following Code Modification:

    at line 252 change '4.1' to '4.0'

At line 255, T'Adjacent (TC_Float,TC_float) /= TC_Float may be True because the function result is given at greater precision for non-model 4.1 than the stored result.

CXB3008, as directed by the AVO, was graded passed by the following Code Modification:

     at line 125, 'atof' was replaced with 'strtod'

The C library function "atof" doesn't have defined semantics when its argument string doesn't fit the model of a numeric value, and for this implementation the test program could be suspended on the call to atof. This code modification imports the C library's strtod function, which has ANSI-defined semantics in this case, thus enabling the test to run as expected.

CXB3009, as directed by the AVO, was graded passed with the following Code Modification:

    comment out lines 264..287

This change simply removes the entire test block beginning at line 264, which checks that Storage_Error is raised as per the standard B.3.1(28). There are many reasons why the expected Storage_Error might not be raised --too much available storage, too little time, even storage reclamation!

CXB3010, as directed by the AVO, was graded passed with the following Code Modification:

  replicate line 199 at line 256, to update the pointer object's value:

      TC_chars_ptr := ICS.New_Char_Array(TC_char_array_2);

The change is necessary to ensure that TC_chars_ptr has a valid pointer value; the original code references TC_chars_ptr after Free was applied to it, and so by B.3.1(51,53) that execution may be erroneous.

BXC6A01, BXC6A02, and BXC6A04, as directed by the AVO, were graded passed with the following Code Modification to the foundation file FXC6A00:

comment out lines 103 & 113

The application of a pragma Volatile to derived types Volatile_Composite and Volatile_Array violates 13.1(10), for these types are untagged derived types (with tagged components) whose parent types are by-reference types (by 6.2:5,8). The only test that references these two types is BXC6A03, and this test is withdrawn (for a similar reason).

CXD1008, as directed by the AVO, was graded passed with the following Code Modification:

comment out the check @228..232

This check may fail if an implementation uses different representations (lengths) of the compared values--one possibly the register contents of evaluation, the other a stored copy--, as the value is not a model number.

CXD6001, as directed by the AVO, was graded passed by the following Code Modification:

at line 114 insert 'with ImpDef;'

and at each of lines 270, 285, & 300 append this delay statement:

'delay ImpDef.Clear_Ready_Queue;'

Because each task of type Victim_Type has the same priority as the main subprogram, they may be suspended while the main subprogram continues to execute a check on their operations--which is not the test's purpose. These code modifications block the main subprogram's execution and thus free the Victim_Type tasks to complete their execution (to the point where they are aborted) before the main subprogram continues and calls procedure Check_Results to check various state variables.

2.3 UNSUPPORTED FEATURES OF THE ADA 95 SPECIALIZED NEEDS ANNEXES

As allowed by [Ada95], an implementation need not support any of the capabilities specified by a Specialized Needs Annex, or it may support some or all of them. For validation testing, each set of tests for a particular Annex is processed only upon customer request, but is processed in full (even if the Ada implementation provides only partial support). When such a test cannot be passed, because the implementation provides only partial support, the result is graded "unsupported" (rather than "inapplicable").

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

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)

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.

CHAPTER 3

PROCESSING INFORMATION

3.1 VALIDATION PROCESS

A full prevalidation was conducted at the AVF's site.

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

A floppy diskette containing the customized test suite (see Section 1.3) was taken on-site by the validation team 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 implementation.

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 validation team. See Appendix A for a complete listing of the processing options for this implementation. It also indicates the default options. No explicit options were used for testing this implementation.

Test output, compiler and linker listings, and job logs were captured on floppy diskette and archived at the AVF.

3.2 MACRO PARAMETERS AND IMPLEMENTATION-SPECIFIC VALUES

This section contains the macro parameters used for customizing the ACVC. 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	4
$COUNT_LAST	2147483647
$ENTRY_ADDRESS	SYSTEM.ITOA(0)
$ENTRY_ADDRESS1	SYSTEM.ITOA(1)
$ENTRY_ADDRESS2	SYSTEM.ITOA(2)
$FIELD_LAST	2147483647
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	75_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1	/NODIRECTORY/FILENAME1
$ILLEGAL_EXTERNAL_FILE_NAME2	/NODIRECTORY/FILENAME2
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1

$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$LESS_THAN_DURATION	-75_000.0
$MACHINE_CODE_STATEMENT	asm'(inst=>"nop");
$MAX_INT	2147483647
$MIN_INT	-2147483648
$NAME	BYTE_INTEGER
$NAME_SPECIFICATION1	X2120A
$NAME_SPECIFICATION2	X2120B
$NAME_SPECIFICATION3	X3119A
$OPTIONAL_DISC	OPTIONAL_DISC
$RECORD_DEFINITION	RECORD inst:string(1..256); END RECORD;
$RECORD_NAME	asm
$TASK_SIZE	128
$TASK_STORAGE_SIZE	1024
$VARIABLE_ADDRESS	FCNDECL.VAR_ADDRESS
$VARIABLE_ADDRESS1	FCNDECL.VAR_ADDRESS1
$VARIABLE_ADDRESS2	FCNDECL.VAR_ADDRESS2

PROCESSING INFORMATION

Package ImpDef and Its Children

The package ImpDef is used by several tests of core language features. Before use in ACVC 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 as it was | the package ImpDef and each of the relevant child packages as they were] modified for this validation. In the interests of simplifying this VSR, the header comment block was removed from [the package file. | each of the package files.]

3.2.1.1 Package ImpDef

```
-- IMPDEF.A
--
--                                Grant of Unlimited Rights
--
--     Under contracts F33600-87-D-0337, F33600-84-D-0280, MDA903-79-C-0687
and
--     F08630-91-C-0015, the U.S. Government obtained unlimited rights in the
--     software and documentation contained herein.  Unlimited rights are
--     defined in DFAR 252.227-7013(a)(19).  By making this public release,
--     the Government intends to confer upon all recipients unlimited rights
--     equal to those held by the Government.  These rights include rights to
--     use, duplicate, release or disclose the released technical data and
--     computer software in whole or in part, in any manner and for any
purpose
--     whatsoever, and to have or permit others to do so.
--
--                                     DISCLAIMER
--
--     ALL MATERIALS OR INFORMATION HEREIN RELEASED, MADE AVAILABLE OR
--     DISCLOSED ARE AS IS.  THE GOVERNMENT MAKES NO EXPRESS OR IMPLIED
--     WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS OF THE
--     SOFTWARE, DOCUMENTATION OR OTHER INFORMATION RELEASED, MADE AVAILABLE
--     OR DISCLOSED, OR THE OWNERSHIP, MERCHANTABILITY, OR FITNESS FOR A
--     PARTICULAR PURPOSE OF SAID MATERIAL.
--*
--
-- DESCRIPTION:
--     This package provides tailorable entities for a particular
--     implementation.  Each entity may be modified to suit the needs
--     of the implementation.  Default values are provided to act as
--     a guide.
--
--     The entities in this package are those which are used in at least
--     one core test. Entities which are used exclusively in tests for
--     annexes C-H are located in annex-specific child units of this package.
--
-- CHANGE HISTORY:
--     12 DEC 93   SAIC    Initial PreRelease version
--     02 DEC 94   SAIC    Second  PreRelease version
```

```
--     16 May 95   SAIC    Added constants specific to tests of the random
--                         number generator.
--     16 May 95   SAIC    Added Max_RPC_Call_Time constant.
--     17 Jul 95   SAIC    Added Non_State_String constant.
--     21 Aug 95   SAIC    Created from existing IMPSPEC.ADA and IMPBODY.ADA
--                         files.
--     30 Oct 95   SAIC    Added external name string constants.
--     24 Jan 96   SAIC    Added alignment constants.
--     29 Jan 96   SAIC    Moved entities not used in core tests into annex-
--                         specific child packages. Adjusted commentary.
--                         Renamed Validating_System_Programming_Annex to
--                         Validating_Annex_C. Added similar
Validating_Annex_?
--                         constants for the other non-core annexes (D-H).
--     01 Mar 96   SAIC    Added external name string constants.
--     21 Mar 96   SAIC    Added external name string constants.
--     02 May 96   SAIC    Removed constants for draft test CXA5014, which was
--                         removed from the tentative ACVC 2.1 suite.
--                         Added constants for use with FXACA00.
--     06 Jun 96   SAIC    Added constants for wide character test files.
--     11 Dec 96   SAIC    Updated constants for wide character test files.
--     13 Dec 96   SAIC    Added Address_Value_IO
--
--!

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

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

   Validating_Annex_G : constant Boolean := True;
   --                                       ^^^^^ --- 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 := 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_1(value: integer) return integer;
--  pragma Import(C, cd30005_1, "CD30005_1");

   CD30005_1_Foreign_Address : constant System.Address:=
               System.Storage_Elements.To_Address ( 16#0000_0000# );
--      cd30005_1'address;
```

```
   --                    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 := 8;
   --                                  ^ --- MODIFY HERE AS NEEDED

   Max_Linker_Alignment  : constant := 8;
   --                                  ^ --- 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/";
     "/x/acvc/acvc21/src/c2/";
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ --- 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.

   --package Address_Value_IO is
   --     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

```
-- IMPDEFC.A
--
--                              Grant of Unlimited Rights
--
--     Under contracts F33600-87-D-0337, F33600-84-D-0280, MDA903-79-C-0687
and
--     F08630-91-C-0015, the U.S. Government obtained unlimited rights in the
--     software and documentation contained herein.  Unlimited rights are
--     defined in DFAR 252.227-7013(a)(19).  By making this public release,
--     the Government intends to confer upon all recipients unlimited rights
--     equal to those held by the Government.  These rights include rights to
--     use, duplicate, release or disclose the released technical data and
--     computer software in whole or in part, in any manner and for any
purpose
--     whatsoever, and to have or permit others to do so.
--
--                                    DISCLAIMER
--
--     ALL MATERIALS OR INFORMATION HEREIN RELEASED, MADE AVAILABLE OR
--     DISCLOSED ARE AS IS.  THE GOVERNMENT MAKES NO EXPRESS OR IMPLIED
--     WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS OF THE
--     SOFTWARE, DOCUMENTATION OR OTHER INFORMATION RELEASED, MADE AVAILABLE
--     OR DISCLOSED, OR THE OWNERSHIP, MERCHANTABILITY, OR FITNESS FOR A
--     PARTICULAR PURPOSE OF SAID MATERIAL.
--*
--
-- DESCRIPTION:
--     This package provides tailorable entities for a particular
--     implementation.  Each entity may be modified to suit the needs
--     of the implementation.  Default values are provided to act as
--     a guide.
--
--     The entities in this package are those which are used exclusively
--     in tests for Annex C (Systems Programming).
--
-- APPLICABILITY CRITERIA:
--     This package is only required for implementations validating the
--     Systems Programming Annex.
--
-- CHANGE HISTORY:
--     29 Jan 96   SAIC    Initial version for ACVC 2.1.
--
--!

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
      16;  -- SIGUSR1
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  --- 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;
   --                               ^^^^ --- 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;
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

   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;
```

3.2.1.3 Package ImpDef.Annex_D

```
-- IMPDEFD.A
--
--                              Grant of Unlimited Rights
--
--     Under contracts F33600-87-D-0337, F33600-84-D-0280, MDA903-79-C-0687
and
--     F08630-91-C-0015, the U.S. Government obtained unlimited rights in the
--     software and documentation contained herein.  Unlimited rights are
--     defined in DFAR 252.227-7013(a)(19).  By making this public release,
--     the Government intends to confer upon all recipients unlimited rights
--     equal to those held by the Government.  These rights include rights to
--     use, duplicate, release or disclose the released technical data and
--     computer software in whole or in part, in any manner and for any
purpose
--     whatsoever, and to have or permit others to do so.
--
```

PROCESSING INFORMATION

```
--                                      DISCLAIMER
--
--     ALL MATERIALS OR INFORMATION HEREIN RELEASED, MADE AVAILABLE OR
--     DISCLOSED ARE AS IS.  THE GOVERNMENT MAKES NO EXPRESS OR IMPLIED
--     WARRANTY AS TO ANY MATTER WHATSOEVER, INCLUDING THE CONDITIONS OF THE
--     SOFTWARE, DOCUMENTATION OR OTHER INFORMATION RELEASED, MADE AVAILABLE
--     OR DISCLOSED, OR THE OWNERSHIP, MERCHANTABILITY, OR FITNESS FOR A
--     PARTICULAR PURPOSE OF SAID MATERIAL.
--*
--
-- DESCRIPTION:
--     This package provides tailorable entities for a particular
--     implementation.  Each entity may be modified to suit the needs
--     of the implementation.  Default values are provided to act as
--     a guide.
--
--     The entities in this package are those which are used exclusively
--     in tests for Annex D (Real-Time Systems).
--
-- APPLICABILITY CRITERIA:
--     This package is only required for implementations validating the
--     Real-Time Systems Annex.
--
-- CHANGE HISTORY:
--     29 Jan 96   SAIC    Initial version for ACVC 2.1.
--
--!

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;
   Maximum_Task_Storage_Size : constant := 400_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;
```

3.3 WITHDRAWN TESTS

At the time of this validation testing, the following 21 tests were withdrawn from the ACVC 2.1 test suite.

Below is the listing of the 21 tests that have been withdrawn from ACVC 2.1 as of 97-10-17. This list supersedes that of 97-09-17, which named 19 tests. Seven of these tests are from the part of ACVC 2.1 that was retained from Ada 83 conformity testing, and so their rationales for withdrawal indicate changes between the two standards.

AMENDED CITATIONS:
C392012 : @350 will raise Constraint_Error when a tag check fails
[This was wrongly listed as "C390012"--which doesn't name a test program.]
LA1001F : has an invalid test objective (cf AI95-00172/01, ARG Minutes 97-04)

ADDED CITATIONS:
none

NEW TESTS:
CD20001 + wrongly requires 'Component_Size to be a factor of word size (13.2:9)
E28002B + @91 the pragma argument violates 10.2.5:6 (no subprogram_declaration)

Line numbers are given relative to the start of a test file in the format of '@<line#,line#,...>'; if a test program comprises more than one file, then the particular file will be indicated by "fn", where "n" is the test-program name's file-designator numeral. E.g., "B38103E : @f1-31, ... " denotes test file b38103e1.ada. In cases where there are two or more rationales for different errors in a test program, these different rationales with their respective line citations are listed on separate lines. References to the Ada 83 & Ada 95 standards are given in the format:

<section|chapter>.<clause>[.<subclause>]:<paragraph>

These references are to the Ada 95 standard unless "Ada83" is specified.

B37312B : @42,58 is legal in Ada95 (3.8.1:7) (vs. RM83 3.7.3:3)

BXC6A03 : @117 applies pragma Volatile in violation of 13.1:10

C390010 : @163 violates 3.10.2:28--(Item.all & Subtype_Parent_Class_Access)

C392010 : @499 the dispatching order produces string "LqmKen", not "NqmNgn"

C392012 : @350 will raise Constraint_Error when a tag check fails

C42006A : @56,66,76 static literals must be rejected (4.9:35)

C48009A : @54 the static expression TA'(0) must be rejected (4.9:35)

C760007 : @161 wrongly expects at least one call to Adjust (7.6:21)

C760012 : @155,183 wrongly requires a component finalization order (7.6.1:9)

C761006 : expects Adjust to be called in Finalize_Test, but it need not be

C761008 : @280 reports Failed but is a possible handler for finalization's P_E

C761009 : @562 has enum.value Task_Requeue_To_Task vice Task_Requeue_To_PO
: @609,651 wrongly expects event Got_Program_Error

C9A005A : @180,192 aborted calls may be yet not cancelled (9.5.3:25, 9.8:15)

C9A008A : @94,191,288 aborted calls may be yet not cancelled (9.5.3:25, 9.8:15)

CD20001 + wrongly requires 'Component_Size to be a factor of word size (13.2:9)

CXC3004 : @282 Attach_Handler raises P_E as per C.3.2:21, which is unintended

CXD4009 : expects a different order of execution than required by priorities

CXD5002 : @138 checks a task priority that may not yet be set (9.8:18, D.5:10)

CXDC001 : @131 doesn't allow for Tasking_Error being raised in activation

E28002B + @91 the pragma argument violates 10.2.5:6 (no subprogram_declaration)

LA1001F : has an invalid test objective (cf AI95-00172/01, ARG Minutes 97-04)

APPENDIX A

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

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

Option	Meaning
Options: (Not all options apply to all languages or targets).	
-a	Generate code for coverage analysis by Multi.
-archive	Create a library archive for use with linker.
-asm=<args>	Pass <args> directly to the assembler.
-B<dir>	Get compiler and other tools from directory <dir>.
-Bdynamic,-Bstatic,-B[no]symbolic	(passed to linker)
-c	Produce object files, but do not link.
-check=none,all,[no]bound,[no]nilderef,[no]variant,[no]switch, [no]assignbound,[no]zerodivide,[no]usevariable.	The compiler generates runtime checks for the items requested.
-cpu=[cpu]	Use specific CPU within family.
-dalign	Always generate double loads and stores.
-fsoft	Use software floating point.
-fnosoft	Use default hardware floating point.
-fnone	In C,C++,Pascal: Give syntax errors for floating point usage
-G	Generate information for MULTI debugger.
-g	Generate information for generic Unix debuggers.
-ga	Force all routines to have a stack frame.
-H	Print names of included headers to stderr.
-help	Print this (abbreviated?) summary.
-Help	Print an even MORE lengthy summary.
-I<dir>	Passed to compiler to add <dir> to include search path.
-K PIC	Position independent code, unlimited symbols. same as -PIC.
-K pic	Position independent code, limited symbols. same as -pic.
-L<dir>	Passed to linker to add <dir> to library search path.
-l<name>	Passed to linker to look for library lib<name>.a.

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

-lnk=<argstring>	Pass <argstring> directly to linker.
-list/<type>	For Ada95, generate the requested listing.
-msg/<msg_kind>	For Ada95, enable message display of msg_kind.
-noautoregister	Only place variables in registers if so declared
-nofloatio	Use printf, scanf without %e %f %g in libansi.
-nomsg/<msg_kind>	For Ada95, suppress message display of msg_kind.
-nooverload	Place each register variable in separate, permanent register.
-nostdlib	Do not add start-up files or libraries to link
-o name	Name final output file.
-OAEILMS	Turn on various optimizations
-Ono???	Turn off a specific optimization. Implies -O. See -Help.
-P	In C, C++, Pascal: preprocess into file.i and stop.
-p	Generate code to profile the executable (prof).
-page/l <length>	For Ada95, set the page length for a source listing.
-page/w <width>	For Ada95, set the page width for a source listing.
-pg	Generate code to profile the executable (gprof).
-pic	Position independent code, limited number of symbols.
-PIC	Position independent code, unlimited symbols.
-passsource	Pass compiler source lines into assembly file.
-Q[yn]	Cause each tool to print its version in its output
-Rdir:dir	Set the runtime path for shared objects
-S	Produce assembly files, and stop.
-sparc2, -sparc10	Generate appropriate code for this workstation
-shared	Produce a shared object instead of an executable.
-src_reg	For Ada95, enable automatic source registration.
-STRICT	In C: require prototypes for functions (implies -strict)
-strict	Generate various compile-time errors and warnings.
-syntax	Compilers will check syntax but not generate code.
-U<name>	In C, C++, Pascal: undefine the macro <name>.
-v	Print all commands before they are executed.
-w	Suppress compiler warning messages.
-W[p02alLCXFP],<arg>[,<arg>]	Pass <arg> to program. (see -Help).
-X<opt>	Pass this option to the compiler.
-Y[p0alSIL],<dir>	Specify alternate directory (see -Help).
-Z<opt>	Pass this option to the compiler.
-#	Print all commands INSTEAD of executing them.

Any unrecognized options are passed to the linker.

APPENDIX B

POINTS OF CONTACT

Ada Validation Facility

Phil Brashear, AVF Manager
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 Validation Organization

Mr. Clyde Roby
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria VA 22311
U.S.A.
Phone   : (703) 845-6666
FAX   : (703) 345-6848
Internet : avo@sw-eng.falls-church.va.us

Ada Joint Program Office

Joan McGarity
Center for Software
Defense Information Systems Agency
5600 Columbia Pike
Falls Church VA 22041
U.S.A.
Phone   : (703) 681-2453
Internet: mcgaritj@ncr.disa.mil

POINTS OF CONTACT

For technical information about this Ada implementation, contact:

Jim Gleason
Green Hills Software, Inc.
2708 Alternate 19 North
Palm Harbor FL 34683
(813) 781-4909
jim@ghs.com

For sales information about this Ada implementation, contact:

Pat Rodenbeck
Green Hills Software, Inc.
2708 Alternate 19 North
Palm Harbor FL 34683
(813) 781-4909
pat@ghs.com

APPENDIX C

REFERENCES

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

[Pro97] Ada Compiler Validation Procedures, Version 5.0, Ada Validation Organization and Ada Joint Program Office, March 1997

[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)