AVF Control Number: EDS19980304RSC05-2.1
DATE COMPLETED
BEFORE ON-SITE: 10 JUN 98
AFTER ON-SITE: 08 JUL 98

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 980613e2.1-023
Rational Software Corporation
RATIONAL APEX ADA 95/83, WINDOWS NT, version 2.0.0
IBM Thinkpad 760XD (pentium) under Windows NT, 4.0

(Final)

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

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 13 June 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 980613e2.1-023 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.

Ada Validation Facility
Phil Brashear, AVF Manager
Electronic Data Systems
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)

Results Summary for 980613e2.1-023

Specialized Needs Annexes

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

SPECIALIZED NEEDS ANNEXES	Total	With-Drawn	Passed	Inappli-cable	Unsup-ported
C Systems Programming & required Section 13 (representation support)	24 161 --- 185	2 1 --- 3	15 160 --- 175	1 0 --- 1	6 0 --- 6
D Real-Time Systems (which requires Annex C)	58	5	0	0	53
E Distributed Systems	26	0	0	0	26
F Information Systems	21	0	0	0	21
G Numerics	29	1	28	0	0
H Safety and Security	30	0	0	0	30

Attachment to VC 980613e2.1-023:
Quantitative Validation Test Results

DECLARATION OF CONFORMANCE

Customer: Rational Software Corporation

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: RATIONAL APEX ADA 95/83, WINDOWS NT version 2.0.0

Host Computer System: IBM Thinkpad 760XD (pentium)
Windows NT, 4.0

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 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 AI95-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

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

C45624A..B (2 tests) check that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types and the results of various floating-point operations lie outside the range of the base type; for this implementation, MACHINE_OVERFLOWS is TRUE.

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.

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

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

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

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

CE2203A checks that WRITE raises USE_ERROR if the capacity of an external sequential file is exceeded; this 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.

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

CE3304A checks that SET_LINE_LENGTH and SET_PAGE_LENGTH raise USE_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this 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. (See section 2.2 re CXB4001.)

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:

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

- Processing Modification: The processing of the test by the Ada 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.

- 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 128 tests (BA21003 is listed twice).

The following 86 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.

B23002A	B23004A	B23004B	B24001A	B24001B
B24001C	B24005A	B24005B	B24007A	B24009B
B24104A	B24204A	B24204B	B24204C	B24204D

IMPLEMENTATION DEPENDENCIES

B24204E	B24204F	B24205A	B24206A	B24206B
B25002A	B25002B	B26001A	B26002A	B26005A
B29001A	B2A003A	B2A003B	B2A003C	B2A003D
B2A003E	B2A003F	B2A005A	B2A005B	B2A007A
B2A021A	B32201A	B33101A	B33201B	B35101A
B36002A	B36201A	B37106A	B38003C	B38003D
B38009D	B393002	B41201A	B44001A	B44004A
B44004B	B44004C	B45205A	B48002A	B48002D
B51001A	B55A01A	B61005A	B67001A	B67001B
B67001C	B67001D	B67001H	B940002	B95001D
B95003A	B95004A	B95007B	B95063A	BA1001D
BA21003	BC1013A	BC1109A	BC1109B	BC1109C
BC1109D	BC1201A	BC1303F	BC2001D	BC2001E
BC3005B	BC3013A	BC51016	BC51017	BC51018
BD4011A				

B393006 and BC51C02, as directed by the AVO, were graded passed with the following code modification:

for B393006, comment out lines 102 & 103;
for BC51C02, comment out line 194

These code modifications will remove unintended illegalities from the test programs, while retaining all intended illegalities (the check that is lost is that compilers don't wrongly treat Func as overriding in cases where it isn't--however, in these cases, it can't be legally declared for the particular checks).

B610001, as directed by the AVO, was graded passed with the following code modification:

comment out lines 221, 223, 225, & 228

These lines are ambiguous, by ARM 3.10.2(2) and 8.6(27).

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.)

B83E01C, B83E01D, and B83E01E, as directed by the AVO, were processed with the following grading modification:

```
the intended illegalities
 for B83E01C at lines 172 & 177 (which are marked with "ERROR")
 for B83E01D at lines 302 & 307 (which are marked with "ERROR")
 for B83E01C, file 3, at lines 56 & 61 (which are marked with "ERROR")
may be regarded instead as ""optional error"s--i.e., they need not
have corresponding diagnostic output from the compiler.
```

Each of these cases is the declaration of a generic subprogram body. The only other modification that would address a compiler's failure to detect these lines as errors would be to split the test with the generic subprogram declarations removed; but this would simply duplicate the cases of the non-generic subprogram bodies, which is checked elsewhere in these test programs.

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.

CA2009C and CA2009F, as directed by the AVO, were graded passed with the following code modification:

```
delete the control-Z characters from each of the test files
```

BA21003, as directed by the AVO, was graded passed with the following processing modification:

```
split the test file BA210030 at line 163, removing the subunit
body of package Bad_Subunit from this otherwise error-free
compilation; process the subunit as a separate compilation.
```

The Ada 95 standard 10.1(4) allows an implementation "to impose implementation-defined restrictions on compilations that contain multiple compilation_units", such as requiring all such units to be error free.

BC3503A, as directed by the AVO, was graded passed with the following code modification:

```
comment out lines 100, 109, & 118  (these lines are LEGAL in Ada 95)
```

Each of the package instantiations PS3, PR3, & PP3 is legal in Ada 95,

as the requirement for matching in Ada 95 is for the formal and actual access TYPES' (not the actual SUBtype's) designated subtypes.

BC3503C, as directed by the AVO, was graded passed with the following code modification:

comment out line 63 (this line is LEGAL in Ada 95)

The package instantiation PU3 is legal in Ada 95 (see BC3503A's entry).

BC51C02, as directed by the AVO, was processed with the following code modification:

comment out line 194

This code modification will remove an unintended illegality from the test program, while retaining all intended illegalities (the check that is lost is that compilers don't wrongly treat Func as overriding in cases where it isn't--however, in this case, it can't be legally declared for the particular check).

CDB0A02, as directed by the AVO, was graded passed with the following code modifications:

at line 313, change the range's upper bound of 24 to '33'; at line 320, change the range's upper bound of 30 to '35'.

move the code at lines 100..102 of FDB0A00 to precede the code at lines 95..98 (i.e., exchange the order of these two sets of code & comments)

The test currently checks for potential overflow based solely on the size of requested storage; but, because alignment adjustments in what storage is allocated might effectively increase the amount of storage "used" in the allocation, the check might fail to account for all used storage. The specified code modification moves the overflow check to follow the calculation for storage (which includes alignment considerations).

CXA5012, as directed by the AVO, was graded passed with the following code modification:

at line 86, change '100_000' to '10_000'

This code modification is necessary for any implementation that defines type Integer to have a 16-bit range.

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.

CXAF001, as directed by the AVO, was graded passed with the following code modification:

    at line 167: change 'Failed' to 'Comment'

CXB3008, as directed by the AVO, was graded passed with the following code modifications:

     at line 106, insert ' type acc_ptr is access IC.char_array; ' at line 107, change function String_To_Double's parameter profile to: '(The_String : in IC.char_array ; End_Ptr: acc_ptr := null)'

     at line 125, change 'atof' to 'strtod'

This code modification alters the previously specified one by giving the Ada function corresponding to strtod a second parameter which, we hope, will make it independent of lunar orientation.

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.

CXB4001, as directed by the AVO, was processed with the following modification and graded inapplicable (see section 2.1):

at line 198: change 'To_Comp' to 'To_Binary'

The function To_Comp was defined in draft versions of the Ada 95 standard but was changed to To_Binary for the final (B.4:45).

CXB4007, as directed by the AVO, was processed with the following modification and graded inapplicable (see section 2.1):

comment out lines 263..268

The Byte_Array values returned by two calls of To_Binary should not be expected to be equal, contrary to this particular check.

CXB5004, as directed by the AVO, was processed with the following modification and graded inapplicable (see section 2.1):

at line f0-79, change 'INVARR(3)' to 'INVARR' [nb: not line 81]

at line f0-83, change 'STR' to 'STR *7'

The changes specified above are necessary in order to produce a legal Fortran program to be used for the test program's interfacing checks.

CXC3001, CXC3003, CXC3006..8, as directed by the AVO, were graded unsupported with the following grading modifications.

Accept the effects of unexpected raising of Program_Error as follows:

for CXC3001, Report.Failed output from lines 286 & 339:
"Unexpected Program_Error in Test_Attach_Handler"
"Unexpected
Program_Error in Test_Exchange_Handler"

for CXC3003/6, program termination with an unhandled exception; for CXC3007/8, Report.Failed output from lines 270/329, resp.:
"Unexpected Program_Error raised"

BXC6001, as directed by the AVO, was graded passed with the following code modification:

comment out pragma Atomic at lines 98, 106, & 116

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).

CXG1004, as directed by the AVO, was graded passed with the following code modification:

at lines 294,307,320,333 replace characters '_i' with '_One'

The required change makes these assignments use the intended variable. The test was coded with a simple typographical error in what are checks of a clearly defined requirement--that Constraint_Error be raised for the complex elementary functions Arctanh & Arccoth with a parameter of plus of minus one. Implementers of the Numerics Annex should understand these requirements regardless of the coding of this ACVC test program.

CXG2002, as directed by the AVO, was graded passed with the following code modification:

at lines 99 & 279 change the expression
'Mre * abs Expected * Real'Model_Epsilon'
to: 'Mre *(abs Expected * Real'Model_Epsilon)'

This change will ensure that the expression is not evaluated by multiplying its two large terms together and overflowing.

CXG2004, as directed by the AVO, was graded passed with the following code modification:

comment out lines 455, 456, & 457 (calls to Sin_Check & Cos_Check)

By removing the calls to the flawed routines, the test program's two other, valid, routines can still be used.

CXG2011, as directed by the AVO, was graded passed with the following code modification:

at line 394: change 'Failed' to 'Comment'

This change allows the non-conforming raising of Argument_Error, and so does not penalize implementers for meeting the test's original requirement. However, implementations should raise Constraint_Error in this case, as per A.5.1(28,29), which will be required under ACVC 2.2 validation.

CXG2012 and CXG2020, as directed by the AVO, were graded passed with the following code modifications:

at lines 99, 124, & 119, respectively of CXG2012 & CXG2020,
change the expression
'Mre * abs Expected * Real'Model_Epsilon'
to: 'Mre *(abs Expected * Real'Model_Epsilon)'

This change will ensure that the expression is not evaluated by multiplying its two large terms together and overflowing.

CXG2013, as directed by the AVO, was graded passed with the following code modification:

comment out line 434 (the call to Special_Angle_Test)

By removing the call to the flawed routine, the test program's other, valid, checks can be made. (The Special_Angle_Test is argued to be too lenient, re Tan with cycle=360.0 degrees, and too severe, re cycle in radians.)

at line 89: change '1000' to '1001'

The change should preclude an even factor of 0.5 in the expression at line 295, and hence the even results of Pi for X and Pi/2 for Y --which is sufficiently near a pole of the Tan function and may overflow (A.5.1:34).

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

comment out line 345 (the call to Subtraction_Error_Test)

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

CXG2016, as directed by the AVO, was graded passed with the following code modification:

comment out lines 417 & 418

These lines contain the only calls to the incorrect procedure Indentity_1_Test. The "conversion to degrees" at line 280 is not sensible, and will wrongly cause the test to be failed.

CXG2017, as directed by the AVO, was graded passed with the following code modifications:

```
change line 212, by inserting parens, from
    X :=  (B - A) * Real (I) / Real (Max_Samples) + A;
 to
    X :=  (B - A) *(Real (I) / Real (Max_Samples))+ A;

comment out line 256 (the first call to Identity_Test)
```

The first code modification removes the potential for overflow, forcing one of the allowed orders of evaluation for the original code. The second change removes the invocation of Identity_Test that checks Tanh values that are too close to zero for the test's error bounds.

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

The following tests for Annex C, Systems Programming, were graded "unsupported":

CXC3001 CXC3003 CXC3005..8

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

CHAPTER 3

PROCESSING INFORMATION

3.1 VALIDATION PROCESS

A partial 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 on the host computer and executed on the target 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. The options invoked explicitly for validation testing during this test were:

-compile X1 X2 X3 ...

This switch is included on the command line. This enables the following actions which would not otherwise occur:

1. Compilation units in each listed file are parsed in order, and if there are duplicate units within the file Xn, the later ones overwrite the earlier ones.

2. If there are syntax errors, a listing file is produced. Syntax corrections that would normally be just applied to the file are instead converted to error messages in the listing file.

3. The compilation units of the file are installed, in an order

chosen by the compiler. If there are semantic errors, a list file is produced. The order of the files in the list file will be in compilation order, not the order in which they appear in the original file.

4. If each of the comp units installed successfully, the compiler searches the view for .ada files that have been obsolesced by the newly compiled files. Any xxx.ada files that are obsolesced are renamed xxx.ada.obs.

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

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	1

$COUNT_LAST	1_000_000_000
$ENTRY_ADDRESS	SYSTEM.STORAGE_ELEMENTS.TO_ADDRESS(30)
$ENTRY_ADDRESS1	SYSTEM.STORAGE_ELEMENTS.TO_ADDRESS(31)
$ENTRY_ADDRESS2	SYSTEM.STORAGE_ELEMENTS.TO_ADDRESS(2)
$FIELD_LAST	2_147_483_647
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	1.0
$ILLEGAL_EXTERNAL_FILE_NAME1	BAD/_CHARACTERS
$ILLEGAL_EXTERNAL_FILE_NAME2	CONTAINS/_WILDCARDS
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$LESS_THAN_DURATION	-1.0
$MACHINE_CODE_STATEMENT	CODE_0'(OP => NOP);
$MAX_INT	2147483647
$MIN_INT	-2147483648
$NAME	SHORT_SHORT_INTEGER
$NAME_SPECIFICATION1	X2120A
$NAME_SPECIFICATION2	X2120B
$NAME_SPECIFICATION3	X3119A
$OPTIONAL_DISC	(OP : OPCODE)
$RECORD_DEFINITION	RECORD OPRND_1 : OPERAND; END RECORD;
$RECORD_NAME	CODE_0
$TASK_SIZE	32
$TASK_STORAGE_SIZE	8192

PROCESSING INFORMATION

$VARIABLE_ADDRESS	FCNDECL.ADDRESS0
$VARIABLE_ADDRESS1	FCNDECL.ADDRESS1
$VARIABLE_ADDRESS2	FCNDECL.ADDRESS2

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 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 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 := 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.
    procedure Cd30005_C_Support;
    pragma Import (C, Cd30005_C_Support, External_Name => "_cd30005_1");
    pragma Link_With ("cd300051.o");

    Cd30005_1_Foreign_Address : constant System.Address :=
       Cd30005_C_Support'Address;

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

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

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

    Cd30005_1_External_Name : constant String := "CD30005_1";
    --                   MODIFY HERE AS NEEDED --- ^^^^^^^^^

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

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

    Max_Default_Alignment : constant := 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/";
    -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ --- 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";
```

```
PROCESSING INFORMATION

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

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

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

    package Address_Value_Io is
       new Ada.Text_Io.Modular_Io (System.Storage_Elements.Integer_Address);

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

end Impdef;


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







package body Impdef is



    -- NOTE: These are example bodies.  It is expected that implementors

    --       will write their own versions of these routines.



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



    -- The time required to execute this procedure must be greater than the

    -- time slice unit on implementations which use time slicing.  For
```

```
    -- implementations which do not use time slicing the body can be null.



    procedure Exceed_Time_Slice is

        T : Integer := 0;

        Loop_Max : constant Integer := 4_000;

    begin

        for I in 1 .. Loop_Max loop

            T := Report.Ident_Int (1) * Report.Ident_Int (2);

        end loop;

    end Exceed_Time_Slice;



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



end Impdef;
```

3.2.1.2 Package ImpDef.Annex_C

```
-- IMPDEFC.A
--
with Ada.Interrupts.Names;

package Impdef.Annex_C is

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

    -- Interrupt_To_Generate should identify a non-reserved interrupt
```

```
    -- that can be predictably generated within a reasonable time interval
    -- (as specified by the constant Wait_For_Interrupt) during testing.

    Interrupt_To_Generate : constant Ada.Interrupts.Interrupt_Id :=
       Ada.Interrupts.Names.Sigbreak;
    -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  --- MODIFY HERE AS NEEDED

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

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

    Wait_For_Interrupt : constant := 0.0;
    --                               ^^^^ --- 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 Text_Io;
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
        Text_Io.Put_Line ("*** Enable_Interrupts: Interrupts not supported");

        -- ^^^^^^^^^^^^^^^^^^^^  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
        Text_Io.Put_Line ("*** Generate_Interrupt: Interrupts not supported");
    end Generate_Interrupt;

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

end Impdef.Annex_C;
```

3.2.1.3 Package ImpDef.Annex_G

```
-- IMPDEFG.A
--
package ImpDef.Annex_G is

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

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

   function Negative_Zero return Float;

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

end ImpDef.Annex_G;
```

PROCESSING INFORMATION

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


package body ImpDef.Annex_G is

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

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

   -- This function must return a negative zero value for implementations
   -- for which Float'Signed_Zeros is True.
   --
   -- The default body simply returns a negated literal 0.0.  If the
   -- default body does not return the value corresponding to a negatively
   -- signed zero for the implementation under test, it must be replaced
   -- by one which does.  See RM A.5.3(13).

   function Negative_Zero return Float is
   begin
      return -0.0;      -- Note: If this value is not negative zero for the
                        --       implementation, use of this "default" value
                        --       could result in false failures in
                        --       implementations where Float'Signed_Zeros
                        --       is True.

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

   end Negative_Zero;

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

end ImpDef.Annex_G;
```

3.3 WITHDRAWN TESTS

At the time of this validation testing, the following 24 tests were withdrawn from the ACVC 2.1 test suite.

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

APPENDIX A

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

A.1 Compilation System Options

The compiler options of this Ada 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.

-compile X1 X2, X3 ...
: Processes the files X1 X2 X3... in order. For each file Xn, the following operations are performed:

  The file Xn can contain several compilation units. Those units are parsed in order, and if there are duplicate units within the file Xn, the later ones overwrite the earlier ones.

  If there are syntax errors, a listing file is produced. Syntax corrections that would normally be just applied to the file are instead converted to error messages in the listing file.

  Next, the compilation units of the file are installed, in an order chosen by the compiler. If there are semantic errors, a list file is produced. The order of the files in the list file will be in compilation order, not the order in which they appear in Xn.

  Finally, if each of the comp units installed successfully, the compiler searches the view for .ada files that have been obsolesced by the newly compiled files. Any xxx.ada files that are obsolesced are renamed xxx.ada.obs.

  By default, these actions are not enabled. Placing the -compile switch on the command line enables these actions.

CLEAN_GOAL
: The state to which the units will be cleaned. All compilation artifacts associated with higher states are deleted.
  Default: Archived

CLOSURE
: Used to determine additional units to be analyzed.

Default: installed

COMPILER_KEY
The compiler to use, which provides for platform-specific semantic checking and code generation.
Default: $APEX_BASE/ada/keys/$APEX_ARCH.2.4.0

FIRST_ERROR
Continue past the first unit with errors. If True, command will stop after the first unit containing an error. Default: False

FLAG_INEVITABLE_EXCEPTIONS
Control the handling of any statically determinable situation that is certain to raise an exception when executed, such as an out-of-bounds assignment. Default: False

IGNORE_INVALID_REP_SPECS
Control the handling of invalid or unsupported representation specifications. Representation specifications are considered invalid if they do not conform to the restrictions specified in "LRM Annex M: Implementation-Dependent Characteristics."
Default: False

IGNORE_REP_SPECS
Ignore representation specifications during semantic analysis.
Default: False

IGNORE_UNSUPPORTED_REP_SPECS
Control the handling of unsupported representation specifications. This switch is overridden by the IGNORE_INVALID_REP_SPECS switch. When IGNORE_INVALID_REP_SPECS is True, this switch has no effect. If IGNORE_INVALID_REP_SPECS is False and this switch is True, unsupported representation specifications are reported with warning messages in the output window and are otherwise ignored. If this switch is False, unsupported representation specifications are treated as errors, causing analysis of the units that contain them to fail. Default: False

INTO
The directory into which the files will be parsed. Must name a view or a directory in a view. If blank, the current directory is used.
Default: " " (current directory)

NEW_RELEASE
Recompiles the units in each view, converting to new DIANA, CG attribute, and program library formats Default: False

OPTIMIZATION_LEVEL
The optimization level to use for compiling. 0 is fastest compilation, 2 is best code. This switch also controls how much inlining is done: at level 0, no routines are inlined; at level 1, only those routines with an applied pragma Inline are candidates for inlining; at level 2, all routines declared within the current same compilation unit as the call site are candidates for inlining, in addition to those made

available at level 1 with a pragma Inline. Default: 0

OPTIMIZATION_OBJECTIVE
The optimization objective, either Time or Space, to be used for any compilation unit that does not contain a pragma Optimize. Execution speed (Time) or code size (Space) are never completely ignored, but if this switch is set to Time, the compiler places greater emphasis on optimizing for speed, while the size of the code is of secondary importance. If it is set to Space, optimizations for speed that increase size are not done. If this switch is set to Space, the loop unrolling optimization (usually performed at level 2) is not performed, Default: Time

PROFILING
The type of profiling to use when preparing the code. Set this switch to "" to turn off profiling. For native compilers, both Gprof and Prof are valid settings. For embedded compilers, only Prof is recognized. Default: ""

REJECT_BAD_LRM_PRAGMAS
Control the handling of illegal Ada pragmas. When True, illegal Ada pragmas are treated as errors, thus causing analysis of the units that contain them to fail. When False, illegal Ada pragmas are reported with warning messages in the output window and are otherwise ignored. Default: False

REJECT_BAD_RATIONAL_PRAGMAS
Control the handling of illegal Rational-defined pragmas. When True, illegal Rational pragmas are treated as errors, thus causing analysis of the units that contain them to fail. When False, illegal Rational pragmas are reported with warning messages in the output window and are otherwise ignored. Default: False

REJECT_INEVITABLE_EXCEPTIONS
Control the handling of any statically determinable situation that is certain to raise an exception when executed, such as an out-of-bounds assignment. When True, this switch overrides the FLAG_INEVITABLE_EXCEPTIONS switch and inevitable exceptions are treated as errors, thus causing analysis of the units that contain them to fail. When False, the treatment of inevitable exceptions depends on the setting of the FLAG_INEVITABLE_EXCEPTIONS switch. Default: False

REJECT_PROMPTS
The compiler will allow you to code units that contain •[statement] prompts. Default: False

REJECT_SYNTAX_ERRORS
The editor and compiler should make syntactic corrections to the programs. When the value is False, corrections are made. When True, corrections are not made; you must make them. Default: False

REJECT_UNDEFINED_PRAGMAS
Control the handling of any pragmas not defined in the LRM or in the

Compiler Reference. When True, undefined pragmas are treated as errors, thus causing analysis of the units that contain them to fail. When False, undefined pragmas are reported with warning messages in the output window and are otherwise ignored. Default: False

TARGET_DIRECTORY
Target directory for cross compiling. Default: " "

TARGET_MACHINE
Host name of the target machine for cross compiling. Default: " "

TRACING
The type of tracing to perform. The value can be a combination of types, e.g. Runtime+Call_Return. Default: RUNTIME

A.2 Linker Options

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

ALL LINKER SWITCHES

ADA_LINK_MODE
: Specifies the link mode for Ada main programs in the view. Valid for targets that support shared libraries. Default: default

COMPILER_KEY
: The compiler to use, which provides for platform-specific semantic checking and code generation.
Default: $APEX_BASE/ada/keys/$APEX_ARCH.2.4.0

CONFIGURATION
: Configuration to use during a link to compute the closure of a main program. If blank, the imports are used. Can be used to specify units to compile beyond those found in the code closure in the Compiler Switches imported views. Default: " "

ELABORATION_ORDER_LISTING
: Create a file containing a listing of the elaboration order of the units in its closure is created when a main program is linked. Elaboration-order listings can be created only for main programs.
Default: False

INCREMENTAL_LINK
: Attempt to use incremental features of the platform linker.
Default: False

LINK_CONTRIBUTION_DEFAULT_MODE
: The LINK_CONTRIBUTION_DEFAULT_MODE switch provides view selective control over linking with/without shared libraries. This switch is only valid when the ADA_LINK_MODE switch in the Ada main program's view is "dynamic_or_static". A LINK_CONTRIBUTION_DEFAULT_MODE of "static" indicates that only object files are to be used from that view in a link process. Any shared library is to be ignored. If the LINK_CONTRIBUTION_DEFAULT_MODE is "dynamic", only the shared library in the view is to be used for a link. It is an error at link-time if the view cannot provide a shared library (that is, the view was compiled with CREATED_SHARED_LIBRARY set to FALSE). A LINK_CONTRIBUTION_DEFAULT_MODE of "dynamic_or_static" specifies that the shared library is to be used if the view is a shared library view (CREATE_SHARED_LIBRARY = TRUE) and the object files are to be used otherwise. If the LINK_CONTRIBUTION_DEFAULT_MODE switch does not have a value in a view, the value defaults to "dynamic_or_static".
Default: dynamic_or_static

NON_ADA_LINKAGE
The arguments to pass to the target linker. This can be used to specify object files and archive libraries for non-Ada program units that will be included when an Ada main program is linked. Default: " "

NONBLOCKING_IO
When a task issues an I/O request, do not wait for the I/O to complete. When True, the task is blocked until the I/O completes, but other tasks in the program may run. When False, the entire program (all tasks) are blocked while any task is waiting for an I/O request to complete.
Default: False

POSIX_COMPLIANT
Use the POSIX I/O behavior. When False,uses traditional I/O behavior allowing users to use some of the UNIX signals that POSIX forbids. Default: True

RUNTIMES
Identifies the directory where archive libraries and objects are found that are used during the link phase.
Default: $APEX_HOME/$APEX_ARCH/lib (native systems), $APEX_BASE/ada/compilers/<target_family>.<version>/ <compiler_variant>/usr(cross systems)

TARGET_DIRECTORY
Target directory. Default: " "

TARGET_MACHINE
Host name of the target machine. Default: " "

USER_LINK_BLOCK
Sets the start address in target memory for linking the first user program. Default: target dependent address

APPENDIX B

POINTS OF CONTACT

Ada Validation Facility

Phil Brashear, AVF Manager
Electronic Data Systems
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 and sales information about this Ada implementation, contact:

Sam Quiring
Rational Software Corporation
1600 NW Compton Drive, Suite 357
Aloha OR 97006
(503) 690-1116 x6732

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)