AVF Control Number: EDS19980609AON01-2.1
DATE COMPLETED
BEFORE ON-SITE: 28 AUG 98
AFTER ON-SITE: 18 SEP 98

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 980901e2.1-036
Aonix
ObjectAda for Unix for SPARC/Solaris, Version 7.1.1
SPARCstation 20 Model 712MP under Solaris, 2.6

(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 1 September 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 980901e2.1-036 which was awarded for this validation is presented on the following page. 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 page.

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 980901e2.1-036

Specialized Needs Annexes

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

SPECIALIZED NEEDS ANNEXES	Total	With-Drawn	Passed	Inappli-cable	Unsup-ported
C Systems Programming	24	*** not processed ***			
& required Section 13 (representation support)	161	x	xxx	x	xxx
	---	---	---	---	---
	185	x	xxx	x	xxx
D Real-Time Systems (which requires Annex C)	58	*** not processed ***			
E Distributed Systems	26	*** not processed ***			
F Information Systems	21	*** not processed ***			
G Numerics	29	*** not processed ***			
H Safety and Security	30	*** not processed ***			

Attachment to VC 980901e2.1-036:
Quantitative Validation Test Results

DECLARATION OF CONFORMANCE

Customer: Aonix

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: ObjectAda for Unix for SPARC/Solaris, Version 7.1.1

Host Computer System: SPARCstation 20 Model 712MP
Solaris, 2.6

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.

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.

BD8001A, BD8002A, BD8003A, BD8004A..C (3 tests), and AD8011A use machine code insertions; this implementation provides a package MACHINE_CODE but does not provide machine code insertions. It provides intrinsic subprograms. (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
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.

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, CXB4007, and CXB4009.)

CXB5001..5 (5 tests) depend upon the availability of an interface to Fortran; this implementation does not support Fortran interfaces. (See Section 2.2 re CXB5004.)

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 56 tests.

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.

B23004A	B24204D	B2A007A	B32201A	B44004C
B55A01A	B830001	BA1101E	BA3006A	BC2001D
BC51017				

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

for B393006, comment out lines 102 & 103; 112..119;
for BC51C02, comment out line 194

These code modifications 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).

C3A2A02, as directed by the AVO, was graded passed with the following code modification:

at line 197, append "pragma Elaborate (C3A2A02_0);"

The library-level instantiation C3A2A02_3 on line 198 can fail elaboration if the body of the generic package C3A2A02_0 is elaborated later than the instantiation.

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).

C760009, as directed by the AVO, was graded passed with the following code modification:

at line 86, add "pragma Elaborate_Body;"

The instantiation C760009_3.Check_1 on line 277 can fail elaboration if the body of the generic package C760009_0 is elaborated later than the instantiation.

C760010, as directed by the AVO, was graded passed with the following code modification:

at line 105, add "pragma Elaborate_Body;"

The library-level instantiation C760010_2 on line 225 can fail elaboration if the body of the generic package C760010_0.Check_Formal_Tagged is elaborated later than the instantiation.

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 are 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.

C9A007A, as directed by the AVO, was graded passed with the following code modifications:

```
    at lines 186 & 217, insert the following delay statements:

      delay ImpDef.Clear_Ready_Queue;
```

This change replaces the Ada83/ACVC 1.11 use of a lower priority in task Register with delay statements, so to ensure that the aborted tasks have time to effect their abortions.

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 BA210031 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.

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.

CA5004B was graded passed by Processing Modification as directed by the AVO. This test checks that an pragma Elaborate is obeyed when it is given for a unit whose body has yet to be compiled or is replaced. However, this implementation doesn't permit a compilation to contain units with the same name, as allowed by [Ada95] 10.1(4). The test file CA5004B0 was split at line 67 into 2 separate files.

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).

CD30002, as directed by the AVO, was graded passed with the following code modifications:

```
  replace lines 75 & 76 with
   type Storage_Element is new System.Storage_Elements.Storage_Element;
   for Storage_Element'Alignment use Impdef.Max_Default_Alignment /4 ;
      --INSERTED ALIGNMENT SPEC TO DOUBLE SIZE OF ARRAY COMPONENT TYPE
  replace lines 128-130 with
   Half_Object    : CD30002_0.O_Half;
   for Half_Object'Alignment
   --    use CD30002_0.S_Units_per_Word * 2;     -- N/A => ERROR.
   use CD30002_0.Multiple_Alignment;     -- AVO CODE MODIFICATION.
```

This implementation does not accept values for alignment that imply a size that is larger than what the implementation supports for the type; an Alignment clause can influence which size the implementation uses. The code modifications above specify a larger-than-default alignment for Storage_Element, which effectively doubles the size of an array of four such components such that the array type can be given the maximum default alignment of eight. The change to use Multiple_Alignment reduces the implied size to an acceptable value (and also brings the alignment clause into agreement with a later check on this value!). At the time of this validation, the ARG had recently tentatively agreed that a compiler need not support alignments greater than the size of the subtype/object (cf AI95-00109/07).

CD30005, as directed by the AVO, was graded passed with the following code modification:

```
    at lines 134 & 148 of test file cd300050,
    change the procedure identifier from 'CD30005' to 'CD300050'.
```

This change will bring the main procedure name into conformity with the ACVC main-unit naming convention (and simplify ACVC processing).

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

BD8001A, BD8002A, BD8003A, BD8004A..C (3 tests), and AD8011A, as directed by the AVO, were graded inapplicable. This implementation provides a package machine code but does not provide machine-code insertions. In this implementation, the compiler does not reject the with clause for package machine_code.

CDB0A02, as directed by the AVO, was graded passed with the following code modification to file FDB0A00:

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

The code 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.

CXA5A01 and CXA5A02, as directed by the AVO, were graded passed with the following code modifications:

```
for CXA5A01:
    append to line 248
      " if New_Float'Machine_Overflows = True then"
    append to line 253 " end if;"
    append to line 264
      " if Float'Machine_Overflows = True then"
    append to line 269 " end if;"

for CXA5A02:
    append to line 258
      " if New_Float'Machine_Overflows = True then"
    append to line 263 " end if;"
    append to line 274
      " if Float'Machine_Overflows = True then"
    append to line 279 " end if;"
```

These changes make certain conformity checks conditional upon the value of 'Machine_Overflows, as specified by [Ada95] clauses A.5.1(28,34) & G.2.4(2,4).

CXA5A03 and CXA5A04, as directed by the AVO, were graded passed with the following code modifications:

```
for CXA5A03:
    insert at line 162
      " if New_Float'Machine_Overflows = True then"
    insert at line 174 " end if;"
    insert at line 310
      " if New_Float'Machine_Overflows = True then"
    insert at line 322 " end if;"
    insert at line 323
      " if Float'Machine_Overflows = True then"
    insert at line 335 " end if;"

for CXA5A04:
    insert at line 103
      " if New_Float'Machine_Overflows = True then"
    insert at line 115 " end if;"
    insert at line 204
      " if New_Float'Machine_Overflows = True then"
    insert at line 237
      " end if; if Float'Machine_Overflows = True then"
    insert at line 251 " end if;"
    insert at line 321
      " if New_Float'Machine_Overflows = True then"
    insert at line 353
      " end if; if Float'Machine_Overflows = True then"
    insert at line 365 " end if;"
```

These changes make certain conformity checks conditional upon the value of 'Machine_Overflows, as specified by [Ada95] clauses A.5.1(28,34) & G.2.4(2,4).

CXA5A09, as directed by the AVO, was graded passed with the following code modifications:

```
insert at line 103
  " if New_Float'Machine_Overflows = True then"
insert at line 115 " end if;"
insert at line 287
  " if New_Float'Machine_Overflows = True then"
insert at line 301 " end if;"
```

These changes make certain conformity checks conditional upon the value of 'Machine_Overflows, as specified by [Ada95] clauses A.5.1(28,34) & G.2.4(2,4).

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

```
at line 167, replace 'Failed' with 'Comment'
```

This change turns a Report.Failed call into a Report.Comment call, which is justified by ARM A.15(16): if the execution environment doesn't support passing arguments to a program, function Command_Name returns the null string (which is what line 167 checks for).

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 imports the C library's strtod function, which has ANSI-defined semantics in the case of a string that doesn't conform to the model for a numeric value and so enables the test program to run as expected. (In some implementations of the C language, function atof will not return the expected value 0.0 in this case; its value is not defined.)

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 graded inapplicable with the following code modification:

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). However, this implementation does not provide the package Interfaces.COBOL and file CXB4001.A is rejected at compile time.

CXB4007, as directed by the AVO, was graded inapplicable with the following code modification:

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. However, this implementation does not provide the package Interfaces.COBOL and file CXB4007.A is rejected at compile time.

CXB4009, as directed by the AVO, was graded inapplicable with the following code modifications:

The COBOL files were replaced by the ACVC's revised files maintained by the certification body. However, this implementation does not provide the package Interfaces.COBOL and file CXB40093.AM is rejected at compile time.

CXB5004, as directed by the AVO, was graded inapplicable with the following code modification:

at line f0-79, change 'INTARR(3)' to 'INTARR' [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. However, this implementation does not provide the package Interfaces.Fortran and file CXB50042.AM is rejected at compile time.

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").

None of the sets of tests for the Specialized Needs Annexes was processed during this validation testing.

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.
The options invoked explicitly for validation testing during this test were:

```
compiler: -v -l -e 250
binder/linker: -nc
```

Test output, compiler and linker listings, and job logs were captured on floppy diskette 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	4
$COUNT_LAST	2147483647
$ENTRY_ADDRESS	FCNDECL.DATA(4)'ADDRESS
$ENTRY_ADDRESS1	FCNDECL.DATA(5)'ADDRESS
$ENTRY_ADDRESS2	FCNDECL.DATA(6)'ADDRESS
$FIELD_LAST	2147483647
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	100_000.0

$ILLEGAL_EXTERNAL_FILE_NAME1	/dev/null/illegal1
$ILLEGAL_EXTERNAL_FILE_NAME2	/dev/null/illegal2
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$LESS_THAN_DURATION	-100_000.0
$MACHINE_CODE_STATEMENT	NULL;
$MAX_INT	2147483647
$MIN_INT	-2147483648
$NAME	SHORT_SHORT_INTEGER
$NAME_SPECIFICATION1	/tmp/X2120A
$NAME_SPECIFICATION2	/tmp/X2120B
$NAME_SPECIFICATION3	/tmp/X3119A
$OPTIONAL_DISC	NO_SUCH_MACHINE_CODE_DISC
$RECORD_DEFINITION	RECORD NULL; END RECORD;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	128
$TASK_STORAGE_SIZE	1024
$VARIABLE_ADDRESS	FCNDECL.DATA(1)'ADDRESS
$VARIABLE_ADDRESS1	FCNDECL.DATA(2)'ADDRESS
$VARIABLE_ADDRESS2	FCNDECL.DATA(3)'ADDRESS

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. In the interests of simplifying this VSR, the header comment block was removed from the package file.

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

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

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

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

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

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

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

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

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

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

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

   Switch_To_New_Task : constant Duration := 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 := 0.5;
--ljn   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 := 0.1;
--ljn   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 := "MY_EXTERNAL_TAG";
-- bh   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.

-- bh
   function CD30005_1_Foreign_Address return System.Address;
   pragma Import( C, CD30005_1_Foreign_Address, "_cd30005_address" );
   --
   -- char * _cd30005_address (void) {
   --    return  _cd30005_1;
   -- }
   --
```

```
-- bh   CD30005_1_Foreign_Address : constant System.Address:=
-- bh                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";
-- bh   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;
-- bh   Max_Default_Alignment : constant := 0;
   --                                   ^ --- MODIFY HERE AS NEEDED

   Max_Linker_Alignment  : constant := 8;
-- bh   Max_Linker_Alignment  : constant := 0;
   --                                   ^ --- 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 --- ^^^^^^^^
```

PROCESSING INFORMATION

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

   -- 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 :=
     "/usr/tests/acvc/acvc2.1/native/tests/c2/";
--ljn     "/data/ftp/public/AdaIC/testing/acvc/95acvc/";
```

```
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ --- MODIFY HERE AS NEEDED

   -- The following two strings must not be modified unless the .AW file
   -- names have been changed.  The Wide Character processor will use
   -- these strings to find the .AW files used in creating the C250001
   -- and C250002 tests.

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

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

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

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

   --M package Address_Value_IO is
   --M      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;

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

```
PROCESSING INFORMATION

end ImpDef;
```

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.

Option	Action	Default
-h(elp	Display a help message.	Opposite
-v(erbose	Output progress messages.	Opposite
-S("assembly_listing"	Generate an assembly listing.	Opposite
-g(enerate_debug	Generate symbolic debugging information.	Opposite
-O1	Enable optimizations which do not substantially interfere with debugging.	No optimization
-O2	Enable all optimizations	No optimization
-a(utoregister	Search and register. Enables automatic registration of source Registration files.	Explicit
-f(ront_end_only	Do not generate code, semantic checking only.	Full Compile
-l(isting	Generate source listing interspersed with	Opposite

	messages/errors.	
-lx(xref_listing	Generate source listing with cross reference information.	Opposite
-s(uppress	Suppress all checks in object Code.	Opposite
-nw(no_warnings	Do not display compiler warnings.	Display Warnings
-nx(no_xref_info	Do not generate cross reference information for use by the browser and cross reference listings.	Opposite
-e(rror_count <n	Stop reporting after <n errors	-e 999
-of <file	Read options and filenames from <file	None
-83	Generate warnings for Ada 95 incompatibilites with Ada 83. Does not generate code, semantic checking only (-f).	Full Compile

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.

Option	Action	Default
-h(elp	Display a help message.	Opposite
-v(erbose	Output progress messages.	Opposite
-S("assembly_listing"	Generate an assembly listing.	Opposite
-g(enerate_debug	Generate symbolic debugging information.	Opposite
-na(no_autoregister	Disables automatic registration of Autoregister source files.	
-nc(no_compile	Do not invoke the compiler to update units.	Invoke Compiler
-cl(compile_locally	Invoke the compiler on inconsistent compile units from linked libraries	Only local units
-e <count	Pass -e <count to the compiler.	Opposite
-p(artition	Invoke the builder to create file ada.o (unless -o is specified) which does not have an Ada main subprogram. Multiple units may be specified. Implies -nl.	Opposite
-nb(no_bind	Do not invoke the binder to an elaboration order. This option generate requires that the elaboration order already exists in the library.	Opposite
-bound_threads	Make all tasks have system wide contentionscope	Opposite
Linker Control Options		
-nl(no_link	Do not invoke the linker; produce Linker link command and relocatable object file main-unit-name.o.	Invoke

-o(utput <file	Put executable code in <file .o With -p put object code in <file .o.	main-unit-name
-Pl(pass_library <lib	Pass the library <lib to the linker library search list.	None
-Po(pass_option <opt	Pass the linker option <opt to the linker.	None
	To pass multiword options, repeat "-Po", i.e., to pass "-c foo" use "-Po -c -Po foo".	
-Pf(pass_object_file <file	Pass the object file <file to the linker.	None

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) 681-2109
Internet: mcgaritj@ncr.disa.mil

POINTS OF CONTACT

For technical and sales information about this Ada implementation, contact:

Aonix
5040 Shoreham Place
San Diego, CA. 92122-5926
(619) 457-2700 or (800) 305-8284

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)