ACAL Control Number: EDS19981105CCC01-2.1
DATE COMPLETED
BEFORE ON-SITE: 09 DEC 98
AFTER ON-SITE: 18 DEC 98

Ada Conformity Assessment Test Report
Certificate Number: A981215E2.1-047
Concurrent Computer Corporation
Concurrent MAXAda 3.1 for PowerPC604
Concurrent Power Hawk 640 (PowerPC 604, Dual Processor)
under PowerMAX OS, 4.2

(Final)

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

TABLE OF CONTENTS

ACAL Control Number: EDS19981105CCC01-2.1

PREFACE

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

The successful completion of conformity assessment is the basis for the issuance of a certificate of conformity and for subsequent registration of related processors. A copy of the certificate A981215E2.1-047 which was awarded for this assessment is presented on the following page. Conformity assessment does not ensure that a processor has no nonconformities to the Ada standard other than those, if any, documented in this report. The compiler vendor declares that the tested processor contains no deliberate deviation from the Ada standard; a copy of this Declaration of Conformity is presented immediately after the certificate.

Base Test Suite Version	ACATS 2.1 (VCS label A2_1C) (See Section 2.2 for details)
Location of Testing	Concurrent Computer Corporation 2101 W. Cypress Creek Rd. Ft. Lauderdale FL 33309
Test Completion Date	15 December 1998

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

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

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

(Insert copy of certificate here)

Results Summary for A981215E2.1-047

Specialized Needs Annexes

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

SPECIALIZED NEEDS ANNEXES	Total	With-Drawn	Passed	Inapplicable	Unsupported
C Systems Programming & required Section 13 (representation support)	24 161 --- 185	2 1 --- 3	22 160 --- 182	0 0 --- 0	0 0 --- 0
D Real-Time Systems (which requires Annex C)	58	5	38	15	0
E Distributed Systems	26	0	---- Not Tested ----		
F Information Systems	21	0	---- Not Tested ----		
G Numerics	29	1	---- Not Tested ----		
H Safety and Security	30	0	---- Not Tested ----		

DECLARATION OF CONFORMITY

Customer: Concurrent Computer Corporation

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

ACATS Version: 2.1

Ada Processor

Ada Compiler Name and Version: Concurrent MAXAda 3.1 for PowerPC604

Host Computer System: Concurrent Power Hawk 640
(PowerPC 604, Dual Processor)
PowerMAX OS, 4.2

Target Computer System: Same as host

Declaration

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

_____ _____

Customer Signature Date

CHAPTER 1

INTRODUCTION

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

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

1.1 USE OF THIS REPORT

Consistent with the national laws of the originating country, the ACAL and ACAA may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). Certified status is awarded only to the processor identified in this report. Copies of this report are available to the public from the ACAL that performed this conformity assessment.

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

1.2 TEST CLASSES

Compliance of Ada processors is tested by means of the ACATS. The ACATS contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and most Class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 contains constants of type SYSTEM.ADDRESS. These constants are used by selected Section 13 tests and by isolated tests for other sections. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for the Input-Output features of the Ada Standard, defined in Annex A of [Ada 95]. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, conformity testing is discontinued.

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

Class L tests check that an Ada processor correctly detects violation of the Ada Standard involving multiple, separately compiled units. In most Class L tests, errors are expected at link time, and execution must not begin. Other L tests may execute and report the appropriate result.

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

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

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

1.3 DEFINITION OF TERMS

Acceptable result	A result that is explicitly allowed by the grading criteria of the test program for a grade of passed or inapplicable.
Ada compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability	The means of checking conformity of Ada processors, consisting of tests, support programs, and a User's Guide. Also referred to as the Ada Conformity Assessment Test Suite.
Ada Conformity Assessment Test Suite (ACATS)	Alternate name for the ACVC (which see).
Ada Conformity Assessment Laboratory	An organization which carries out the procedures required to assess the conformity of an Ada processor.
Ada Conformity Assessment Authority (ACAA)	The organization that provides coordination and technical guidance for the Ada Conformity Assessment Laboratories.
Ada	An Ada processor.
Certified Status	(Also "certified as conforming") The status granted to an Ada processor by the award of an Ada Conformity Assessment Certificate.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-wriiten or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfillment by a product, process or service of all requirements specified.
Conformity Assessment	The process of checking the conformity of an Ada processor to the Ada programming language and of issing a certificate for that processor.

Customer	An individual or corporate entity who enters into an agreement with an ACAL which specifies the terms and conditions for ACAL services (of any kind) to be performed.
Declaration of Conformanity	A formal statement from a customer assuring that conformity is realized or is attainable on the Ada processor for which certified status is realized.
Foundation Unit (Foundation Code)	An Ada package used by multiple tests. Foundation units are designed to be reusable. A valid foundation unit must be in the Ada library for those tests that are dependent on the foundation unit.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable Test	A test that contains one or more test objectives found to be irrelevant for the given Ada processor.
ISO	International Organization for Standardization.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management.
Specialized Needs Annex	One of annexes C through H of [Ada95]. Testing of one or more specialized needs annexes is optional, and results for each tested annex are summarized in this report.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Unsupported Feature Test	A test for a language feature that is not required to be supported, because it is based upon a requirement stated in an Ada 95 Specialized Needs Annex.
Withdrawn Test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 INAPPLICABLE TESTS

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

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

C45536A contains length clauses that specify values for 'SMALL that are not powers of two or ten; this processor does not support such values for 'SMALL.

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 processor, MACHINE_OVERFLOWS is TRUE.

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

BC50004 contains length clauses that specify values for 'SMALL that are not powers of two and depends upon support of Decimal Fixed Point Types; this processor does not support such values for 'SMALL nor does it support Decimal Fixed Point Types. This test is rejected for these reasons at compile time.

CC50A01 was graded inapplicable for this processor as the result of a dispute resolution by the ACAA. (See Section 2.2.)

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

CD2A53A checks operations of a fixed-point type for which a length clause specifies a power-of-ten TYPE'SMALL; this processor does not support decimal 'SMALLs. (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 processor supports these operations.

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

CE2203A checks that WRITE raises USE_ERROR if the capacity of an external sequential file is exceeded; this processor cannot restrict file capacity.

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

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

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

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

CXAA010 depends upon support of Decimal Fixed Point Types. This implementation does not support Decimal Fixed Point Types and the test is rejected at compile time.

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

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

CXD1006 requires that an implementation support tasks running at interrupt priorities. This implementation does not support tasks running at interrupt priorities, as allowed by D.2.1(10). The offending pragmas are rejected at compile-time. (See Section 2.2.)

CXD2001..4 (4 tests), CXD2008, and CXD6001..3 (3 tests) test objectives that are valid only for uni-processor implementations. This implementation has multiple processors. (See Section 2.2 re CXD2004, CXD6001, and CXD6002.)

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

2.2 MODIFICATIONS

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

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

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

- Evaluation Modification: The evaluation of a test result is changed.

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

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

Modifications were required for 47 tests.

The following 5 tests were split into two or more tests because this processor did not report the violations of the Ada Standard in the way expected by the original tests.

B32201A	B36201A	B490002	BA1101B	BC51016

The following 39 allowable test modifications from the "ACATS Modifications List", Version 2.1C were used.

B392002	B490001	B83E01C	B83E01D	B83E01E
BA21003	BDE0001	BXC6A02	BXD1001	C330001
C332001	C3A2A02	C460008	C760009	C760010
C9A007A	CD10001	CD2A53A	CD30002	CD30005
CDE0001	CE3115A	CXA5012	CXA5015	CXA5A01
CXA5A02	CXA5A03	CXA5A04	CXA5A09	CXAF001
CXB3008	CXC7001	CXD1006	CXD1008	CXD2004
CXD6001	CXD6002	CXD8002	FDB0A00	

For tests BA3006A and BA3006B, the ACAA has ruled that these tests can be declared Passed by Evaluation Modification for this implementation. The intent of the tests are to determine that partition consistency is preserved when a unit is substantially modified. This goal is met by virtually any error messages in the tests.

For test BA3006A, if errors appear in previously compiled units when the unit BA3006A5 is compiled (or later), the test is passed.

For test BA3006B, if errors appear in previously compiled units when the unit BA3006B2 is compiled (or later), the test is passed.

For test CC50A01, the ACAA has ruled that this test can be declared Not Applicable by Evaluation Modification for this implementation.

The petitioner argues that the Ada Reference Manual does not allow the matching of a formal object of an enclosing package to the actual parameter of the actual instantiation given for a formal package. The experts consulted by the ACAA agree with this argument.

However, the ACAA encourages the petitioner to modify their processor to pass the test. The test in question was in ACVC 2.0.1, and at least some implementations passed it then, and all 46 conformity assessments done to date with ACATS 2.1 passed the test. The experts consulted by the ACAA believe that there is a strong probability that the ARG will rule to make this test legal, and have given a high priority to this issue. Finally, the ACAA believes, and the experts concur, that the construct in question is a natural way to use generic formal packages, and it is quite likely to appear in user code.

2.3 UNSUPPORTED FEATURES OF THE ADA 95 SPECIALIZED NEEDS ANNEXES

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

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

Annex E, Distributed Systems (all BXE* & CXE* files)
Annex F, Information Systems (all BXF* & CXF* files)
Annex G, Numerics (all CXG* files)
Annex H, Safety and Security (all BXH*, CXH*, & LXH* files)

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

No tests for Annex D, Real-Time Systems, were graded "unsupported".

The implementer has identified the following Specialized Needs Annex features that are not supported by this implementation, but are not detected by any of the tests:

Feature	RM Reference
Recommended representation support for the following clauses:	C.2
- 13.1(22) -- support of non-static constant expressions	
- 13.3(19) -- page alignment of standalone library-level objects	

- 13.3(35) -- inhibit optimizations based on assumptions of no aliases

Preelaboration requirements	C.4
Atomic objects are not always moved indivisibly	C.6(15)
Not all storage associated with attributes of a task is reclaimed upon task termination.	C.7.2(17)
Not all storage associated with a task is reclaimed when a designated task is freed via Ada.Unchecked_Deallocation.	D.12(4)

CHAPTER 3

PROCESSING INFORMATION

3.1 CONFORMITY ASSESSMENT PROCESS

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

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

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

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

The tests were compiled, linked, and executed on the host computer system.

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

PROCESSING INFORMATION

The following explicit option settings were used during witness testing:

Compiler option/switch		Effect
[-el]	(errors list)	Source and errors to stdout
[-w]	(warnings)	Suppress warning and info messages
[-g]	(debug)	Select debug level : full

Linker option/switch		Effect
[-o]	(output)	Override the default file for the partition
[-w]	(warnings)	Suppress warning messages

a.build option/switch		Effect
[-attempt]	(force attempts)	Attempt those compilations and links that will fail, but skip subsequent dependent compilations and links.
[-source file]	(source file)	Build all units defined in the given source "file" and all units upon which they directly or indirectly depend.

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

3.2 MACRO PARAMETERS AND IMPLEMENTATION-SPECIFIC VALUES

This section contains the macro parameters used for customizing the ACATS. The meaning and purpose of these parameters are explained in [UG97]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN, also listed here. These values are expressed in a symbolic notation, using placeholders as appropriate.

3.2.1 Macro Parameters

Macro Parameter	Macro Value
$MAX_IN_LEN	499
$BIG_ID1	AAA ... A1 (499 characters)
$BIG_ID2	AAA ... A2 (499 characters)
$BIG_ID3	AAA ... A3A ... A (499 characters)
$BIG_ID4	AAA ... A4A ... A (499 characters)
$BIG_STRING1	"AAA ... A" (499/2 characters)
$BIG_STRING2	"AAA ... A1" ((499/2)-1 characters)
$BLANKS	" ... " (499-20 blanks)
$MAX_STRING_LITERAL	"AAA ... A" (499 characters)
$ACC_SIZE	32
$ALIGNMENT	8
$COUNT_LAST	2_147_483_647
$ENTRY_ADDRESS	virtual_address(16#40#)
$ENTRY_ADDRESS1	virtual_address(16#80#)
$ENTRY_ADDRESS2	virtual_address(16#100#)
$FIELD_LAST	2_147_483_647
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	100_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1	no/such/file/name
$ILLEGAL_EXTERNAL_FILE_NAME2	/this/file/does/not/exist
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INTEGER_FIRST	-2147483648

PROCESSING INFORMATION

$INTEGER_LAST	2147483647
$LESS_THAN_DURATION	-75_000.0
$MACHINE_CODE_STATEMENT	code_1'(mftbl, r7);
$MAX_INT	2147483647
$MIN_INT	-2147483648
$NAME	TINY_INTEGER
$NAME_SPECIFICATION1	/hen2/imran/acvc2.1_phase3_0132/EXEC/ppc604/tests/ce/X2120A
$NAME_SPECIFICATION2	/hen2/imran/acvc2.1_phase3_0132/EXEC/ppc604/tests/ce/X2120B
$NAME_SPECIFICATION3	/hen2/imran/acvc2.1_phase3_0132/EXEC/ppc604/tests/ce/X3119A
$OPTIONAL_DISC	(OP:OPCODE)
$RECORD_DEFINITION	RECORD OPRND_1: OPERAND ; END RECORD ;
$RECORD_NAME	code_1
$TASK_SIZE	32
$TASK_STORAGE_SIZE	10240
$VARIABLE_ADDRESS	FCNDECL.SPACE(1024)
$VARIABLE_ADDRESS1	FCNDECL.SPACE(1024)
$VARIABLE_ADDRESS2	FCNDECL.SPACE(1024)

Package ImpDef and Its Children

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

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

3.2.1.1 Package ImpDef

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

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

package ImpDef is

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

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

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

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

   Validating_Annex_E : constant Boolean := 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 := 5.0;
   --                                       ^^^ --- MODIFY HERE AS NEEDED

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

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

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

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

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

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

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

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

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

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

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

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

   procedure Exceed_Time_Slice;

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

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

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

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

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

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

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

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

   function cd30005_addr return System.Storage_Elements.Integer_Address;
   pragma Import(C, cd30005_addr);
   pragma linker_options("-ld ../support/cd30005_addr.o " &
                         "-ld ../support/cd300051.o");

   CD30005_1_Foreign_Address : constant System.Address:=
               System.Storage_Elements.To_Address ( cd30005_addr );
   --                   MODIFY HERE AS REQUIRED --- ^^^^^^^^^^^^
```

```
PROCESSING INFORMATION


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

   -- 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 := 16 ;
   --                                  ^ --- MODIFY HERE AS NEEDED

   Max_Linker_Alignment  : constant := 16 ;
   --                                  ^ --- MODIFY HERE AS NEEDED

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

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

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

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

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

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

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

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

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

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

   -- The following string constants must be the external names resulting
   -- from the Fortran compilation of CXB50040.FTN, CXB50041.FTN,
   -- CXB50050.FTN, and CXB50051.FTN.
   --
```

```
   -- The strings will be used as arguments to pragma Import.
   --
   -- Note that the use of these four string constants will be split between
   -- two tests, CXB5004 and CXB5005.

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

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

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

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

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

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

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

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

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

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

   Test_Path_Root : constant String :=
     "../c2/";
--   "/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.
```

```
PROCESSING INFORMATION


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

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

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

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

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

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

end ImpDef;


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


package body ImpDef is

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

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

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

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

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

end ImpDef;
```

3.2.1.2 Package ImpDef.Annex_C

```
-- IMPDEFC.A
--
--!

with Ada.Interrupts.Names;
with posix_1003_1 ;

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.sigrt1 ;
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  --- MODIFY HERE AS NEEDED

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

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

   Wait_For_Interrupt : constant := 10.0;
   --                              ^^^^ --- MODIFY HERE AS NEEDED

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

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

   procedure Enable_Interrupts;

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

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

   procedure Generate_Interrupt;

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

end ImpDef.Annex_C;


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

```
package body ImpDef.Annex_C is

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

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

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

   procedure Enable_Interrupts is
   begin
      null;

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

   end Enable_Interrupts;

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

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

   procedure Generate_Interrupt is
  void : integer ;
   begin
      Report.Comment (". >>>>> GENERATE THE INTERRUPT NOW <<<<< ");
  void := posix_1003_1.kill (posix_1003_1.getpid, posix_1003_1.sigrt1) ;

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

   end Generate_Interrupt;

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

end ImpDef.Annex_C;
```

3.2.1.3 Package ImpDef.Annex_D

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

package ImpDef.Annex_D is

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

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

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

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

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

   type Processor_Type is (Uni_Processor, Time_Slice, Multi_Processor);

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

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

end ImpDef.Annex_D;
```

3.3 WITHDRAWN TESTS

At the time of this conformity assessment testing, the following 25 tests were withdrawn from the ACATS.

B37312B	BXC6A03	C390010	C392010	C392012	C42006A
C48009A	C760007	C760012	C761006	C761008	C761009
C9A005A	C9A008A	CD20001	CXC3004	CXD2005	CXD4009
CXD5002	CXDB005	CXDC001	CXG2022	E28002B	EA3004G
LA1001F					

APPENDIX A

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

A.1 Compilation System Options

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

==========

a.compile options

==========

Compile Options:

-e[e|l|v|L]	(errors)	Control error emission style: -e erroneous lines and errors to stdout -el source and errors to stdout -eL source and errors to stdout, even when no errors are present -ev errors to source and calls vi -ee errors to source and calls $EDITOR (default: errors only to stderr)
-g[level]	(debug)	Select debug level: 0 (none), 1 (lines) or 2 (full); -g is equivalent to -g2
-i	(info)	Suppress information only messages
-opp	(opportunism)	Make opportunistic use of unit bodies (beyond inlining); requires additional dependencies
-sm share_mode	(share_mode)	Apply pragma SHARE_MODE(share_mode)
-w	(warnings)	Suppress warning and info messages
-N	(not shared)	Set default of pragma SHARE_BODY to FALSE
-O[level]	(optimize)	Select level of code optimization (1-3)
-Qkeyword[=value]	(qualifier)	Specify a qualifier keyword (-HQ for list)
-S	(suppress)	Suppress checks (same as pragma SUPPRESS_ALL)

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

Compile options may be preceded by "!" to negate them (e.g. "-!S")

A.2 Linker Options

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

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

a.link options

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

Link Options:

  -ar=lx          (archive)         Pass a -lx option to the system loader (ld),
                                    ensuring that it will be statically linked
  -bound          (bound)           Set the default task weight to BOUND
  -c[v|V]         (compress)        Compress generated program via selective
                                    linking; remove "dead" routines
  -elab sym       (elaborate)       Insert the routine specified by "sym" as the
                                    first elaboration routine
  -elab_src       (elab source)     Create a source file named ".ELAB_{main}.a"
                                    representing the elaboration of library
                                    units and execution of the main subprogram
  -forgive        (forgive)         Cause a partition to be linked despite the
                                    fact that some of its units are either not
                                    compiled or inconsistent
  -ld file        (ld file)         Pass an object file to the system loader
                                    (ld)
  -multiplexed    (multiplexed)     Set the default task weight to MULTIPLEXED
  -nosoclosure    (no so closure)   Do not include full transitive closure of
                                    shared objects' units
  -ntrace         (NightTrace)      Use an "ntrace" runtime which requires
                                    the NightTrace daemon for execution
  -skipobscurity (skip obscurity)   Skip obscurity checks
  -so=lx          (shared object)   Pass a -lx option to the system loader (ld),
                                    ensuring that it will be dynamically linked
  -trace          (trace)           Use a "trace" version of the runtime
  -A args         (a.analyze)       Use "args" as list of arguments to a.analyze
                                    (spaces must be quoted with "\")
  -O              (optimize)        Perform optimizations at link time by
                                    invoking the a.analyze optimizer
```

A.3 Build Options

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

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

a.build options

==========================================================================
Summary: Compile and link as necessary to build a unit, partition or
environment

Usage: a.build [options] [partitions ...]

Options:

  -allparts       (all partitions)    Build all partitions in the environment
  -attempt        (force attempts)    Attempt those compilations and links
                                      that will fail, but skip subsequent
                                      dependent compilations and links
  -bypass         (bypass optional)   Bypasses optional dependencies if they
                                      cannot be satisfied by the build
  -env env        (environment)       Specify an environment pathname
  -i              (infos)             Suppress a.build infos
  -noimport       (no import)         Forestall automatic recompilation of
                                      out-of-date units from other
                                      environments in the current environment
  -nosource       (no source)         Skip checks of the source timestamps for
                                      out-of-date units (should only be used
                                      if no source files have changed)
  -o file         (output)            Override the output file for the
                                      partition being built
  -part partition (partition)         Build the given "partition", all
                                      included units and all units upon which
                                      they directly or indirectly depend
  -p[n]           (parallel)          Do up to "n" parallel compilation ("n"
                                      defaults to number of CPUs)
  -pd[n]          (parallel deps)     Do up to "n" parallel dependency
                                      analyses ("n" defaults to number of
                                      CPUs * 2)
  -r unit         (require)           Build the given "unit", all units upon
                                      which it directly or indirectly depends,
                                      and all units which directly or
                                      indirectly depend upon it
  -rfile file     (require from file) Build the units in "file", all units
                                      upon which they directly or indirectly
                                      depend, and all units which directly or
                                      indirectly depend upon them
  -rel release    (release)           Specify a release
  -source file    (source file)       Build all units defined in the given
                                      source "file" and all units upon which
                                      they directly or indirectly depend
  -state s        (state)             Build all specified units to compilation
```

		state "s" (compiled is used by default)
-stop	(stop on errors)	If an error is encountered, stop building (normally, any units not dependent upon the erroneous units would be built)
-u unit	(unit)	Build the given "unit" and all units upon which it directly or indirectly depends
-ufile file	(units from file)	Build the units in "file" and all units upon which they directly or indirectly depend
-v	(verbose)	Display compilations as they are done
-vv	(very verbose)	Display commands as they are done
-w	(warnings)	Suppress a.build warnings
-Attempt	(force attempts!)	Attempt those compilations and links that will fail, including subsequent dependent compilations and links
-C "compiler"	(compiler)	Use "compiler" to compile units (may be used to pass options to the compiler, e.g. a.build -C "a.compile -b")
-H	(help)	Display this description and stop
-IO[level]	(interoptimization)	Set level of interoptimization (0-1)
-L "linker"	(linker)	Use "linker" to link partitions (may be used to pass options to the linker)
-V	(verify)	List compilations that would occur, but do not actually perform them

Specified partitions are equivalent to partitions passed as arguments to the "-part" option. If no options or partitions are specified, then all units and partitions in the environment are built.

The following pages contain a sample test script and a short description of that script.

==========

testing script

==========

The following script is an sample script that would process a single test (in this case, c41303f).

```
# Setup an environment. This script presumes that the
# support environment has already been built in ../support.


a.mkenv
a.path -a obsolescent
a.path -a ../support

###  Compile, Link, and Execute a Test

# Introduce units to the environment

a.intro c41303f.ada > /tmp/errors 2>&1

# Resolve any ambiguities via the resolve script (supplied)

resolve_script  /tmp/errors

# Create the partition for the test

a.partition -create active c41303f

# Compile and link the test

a.build c41303f

# Run the test

./C41303F
```

The following "resolve" script parses the output from the a.intro tool. If an error from a.intro indicating that potentially ambiguous unit(s) have been introduced, the ambiguity is resolved in favor of the most recently introduced unit(s). Error checking is done to detect erroneous input and errors from the a.resolve tool. If no ambiguities are detected, no action is taken.

The script expects a single argument, which is a file containing the output (stdout and stderr) from an invocation of a.intro.

The steps in the script are:

1. Parse the a.intro output for errors indicating ambiguous units.
   (a.intro is called on each source file. Its purpose is to
   introduce units in that source file to the environment)

2. For each error detected:

   Extract the spec/body information, unit name and file from the error.

   Validate that information, emitting an error if it is inconsistent
   with the information expected.

   Resolve the unit (via a.resolve) by selecting the unit from the current
   source file.

resolve script

```
#!/bin/ksh

result_file=${1}

if [ -z "${result_file}" ]; then
   echo Usage: ${0} result_file
   exit 1
fi

grep "already exists in source file" < $result_file |
sed "s/\" already exists in.*//;s/.* .*: //;s/ from \"/ /;s/\"//;s/of //" |
sort -u |
while read portion unit file; do

   case "${portion}" in
   spec) true ;;
   body) true ;;
   *)    echo "${0}: error parsing ambiguity output (portion) from a.intro" ;
         exit 1 ;;
   esac ;
   if [ -z "${unit}" ]; then
      echo "${0}: error parsing ambiguity output (unit) from a.intro" ;
      exit 1 ;
   fi

   if [ -z "${file}" -o ! -r "${file}" ]; then
      echo ${0}: error: cannot resolve ambiguity for ${unit}/${portion}
   fi

   if a.resolve -r ${file} ${unit}/${portion}; then
      true ;
   else
      echo "${0}: error resolving ${unit}/${portion} to ${file}" ;
      exit 1 ;
   fi
done
```

APPENDIX B

POINTS OF CONTACT

Ada Conformace Assessment Laboratory

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

AdaConformity Assessment Authority

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

POINTS OF CONTACT

For technical information about this Ada processor, contact:

Ed Kelly
Concurrent Computer Corporation
2101 W. Cypress Creek Rd.
Ft Lauderdale FL 33309
(954) 973-5340

For sales information about this Ada processor, contact:

Rob Menzel
Concurrent Computer Corporation
2101 W. Cypress Creek Rd.
Ft Lauderdale FL 33309
(800) 666-4544

APPENDIX C

REFERENCES

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

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

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

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

REFERENCES

end of document

(REMOVE THIS PAGE)