

# REPORT DOCUMENTATION PAGE

Form Approved  
OPM No.

2

## AD-A262 961



estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering information, and completing the review. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Project Director, Washington, DC 20503.

2. REPORT	3. REPORT TYPE AND DATES Final: 30 April 1992
-----------	--

4. TITLE AND Validation Summary Report: TLD Systems, Ltd., TLD MV/MV Ada Compiler System, Version 2.9.0, Data General MV/32 2000-2 under AOS/VS II, Version 2.03 (Host & Target), 920319W1.11238	5. FUNDING
---	------------

6. Wright-Patterson AFB, Dayton, OH USA
---

7. PERFORMING ORGANIZATION NAME(S) AND Ada Validation Facility, Language Control Facility ASD/SCEL Bldg. 676, Rm 135 Wright-Patterson AFB, Dayton, OH 45433
--

8. PERFORMING ORGANIZATION AVF-VSR-523-0392
--

9. SPONSORING/MONITORING AGENCY NAME(S) AND Ada Joint Program Office United States Department of Defense Pentagon, Rm 3E114 Washington, D.C. 20301-3081
---

10. SPONSORING/MONITORING AGENCY
----------------------------------

11. SUPPLEMENTARY
-------------------

**DTIC**  
**SELECTED**  
**MAR 31 1993**  
**S B D**

12a. DISTRIBUTION/AVAILABILITY Approved for public release; distribution unlimited.
--

12b. DISTRIBUTION
-------------------

13. (Maximum 200) TLD Systems, Ltd., TLD MV/MV Ada Compiler System, Version 2.9.0, Data General MV/32 2000-2 under AOS/VS II, Version 2.03 (Host & Target), ACVC 1.11.
---

98 3 30 067

93-06530



14. SUBJECT Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A,
--

15. NUMBER OF
---------------

16. PRICE
-----------

17. SECURITY CLASSIFICATION UNCLASSIFIED
---

18. SECURITY UNCLASSIFIED
------------------------------

19. SECURITY CLASSIFICATION UNCLASSIFIED
---

20. LIMITATION OF
-------------------

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 19 March 1992.

Compiler Name and Version: TLD MV/MV Ada Compiler System,  
Version 2.9.0

Host Computer System: Data General MV/32 20000-2,  
under AOS/VS II, Version 2.03

Target Computer System: Data General MV/32 20000-2,  
under AOS/VS II, Version 2.03

Customer Agreement Number: 91-11-14-TLD


See section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 920319W1.11238 is awarded to TLD Systems, Ltd. This certificate expires on 1 June 1993.


This report has been reviewed and is approved.



Ada Validation Facility  
Steven P. Wilson  
Technical Director  
ASD/SCEL  
Wright-Patterson AFB OH 45433-6503



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



Ada Joint Program Office  
Dr. John Solomond, Director  
Department of Defense  
Washington DC 20301

AVF Control Number: AVF-VSR-523-0392  
30 April 1992  
91-11-14-TLD

Ada COMPILER  
VALIDATION SUMMARY REPORT:  
Certificate Number: 920319W1.11238  
TLD Systems, Ltd.  
TLD MV/MV Ada Compiler System, Version 2.9.0  
Data General MV/32 20000-2 under ACS/VS II, Version 2.03 =>  
Data General MV/32 20000-2 under AOS/VS II, Version 2.03

Prepared By:  
Ada Validation Facility  
ASD/SCEL  
Wright-Patterson AFB OH 45433-6503

DTIC QUALITY INSPECTED

Mission For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

DECLARATION OF CONFORMANCE

Customer: TLD Systems, Ltd.

Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB, OH 45433-6503

ACVC Version: 1.11

Ada Implementation:


Compiler Name and Version: TLD MV/MV Ada Compiler System, Version 2.9.0

Host Computer System: Data General MV/32 20000-2,  
AOS/VS II, Version 2.03

Target Computer System: Data General MV/32 20000-2,  
AOS/VS II, Version 2.03

Customer's Declaration

I, the undersigned, representing TLD Systems, Ltd., declare that TLD Systems, Ltd. has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration executing in the default mode. The certificates shall be awarded in TLD Systems, Ltd.'s corporate name.

  
\_\_\_\_\_  
TLD Systems, Ltd.  
Terry L. Dunbar, President

Date: 29 November 1991

## TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	USE OF THIS VALIDATION SUMMARY REPORT . . . . .	1-1
1.2	REFERENCES. . . . .	1-2
1.3	ACVC TEST CLASSES . . . . .	1-2
1.4	DEFINITION OF TERMS . . . . .	1-3
CHAPTER 2	IMPLEMENTATION DEPENDENCIES	
2.1	WITHDRAWN TESTS . . . . .	2-1
2.2	INAPPLICABLE TESTS. . . . .	2-1
2.3	TEST MODIFICATIONS. . . . .	2-4
CHAPTER 3	PROCESSING INFORMATION	
3.1	TESTING ENVIRONMENT . . . . .	3-1
3.2	SUMMARY OF TEST RESULTS . . . . .	3-1
3.3	TEST EXECUTION. . . . .	3-2
APPENDIX A	MACRO PARAMETERS	
APPENDIX B	COMPILATION SYSTEM OPTIONS	
APPENDIX C	APPENDIX F OF THE Ada STANDARD	

## CHAPTER 1

### INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). 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 [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

#### 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). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service  
5285 Port Royal Road  
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization  
Computer and Software Engineering Division  
Institute for Defense Analyses  
1801 North Beauregard Street  
Alexandria VA 22311-1772

## INTRODUCTION

### 1.2 REFERENCES

- [Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
- [Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.
- [UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

### 1.3 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 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 SPRT13, 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 SPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. 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. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values — for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional 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 2.3.

For 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 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

#### 1.4 DEFINITION OF TERMS

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.
Ada Implementation	An Ada compiler 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.
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.

## INTRODUCTION

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 attainable on the Ada implementation for which validation status is realized.
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.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>.<paragraph>."
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
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 [Pro90].
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 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 2 August 1991.

E28005C	B28006C	C32203A	C34006D	C35508I	C35508J
C35508M	C35508N	C35702A	C35702B	B41308B	C43004A
C45114A	C45346A	C45612A	C45612B	C45612C	C45651A
C46022A	B49008A	B49008B	A74006A	C74308A	B83022B
B83022H	B83025B	B83025D	C83026A	B83026B	C83041A
B85001L	C86001F	C94021A	C97116A	C98003B	BA2011A
CB7001A	CB7001B	CB7004A	CC1223A	BC1226A	CC1226B
BC3009B	BD1B02B	BD1B06A	AD1B08A	BD2A02A	CD2A21E
CD2A23E	CD2A32A	CD2A41A	CD2A41E	CD2A87A	CD2B15C
BD3006A	BD4008A	CD4022A	CD4022D	CD4024B	CD4024C
CD4024D	CD4031A	CD4051D	CD5111A	CD7004C	ED7005D
CD7005E	AD7006A	CD7006E	AD7201A	AD7201E	CD7204B
AD7206A	BD8002A	BD8004C	CD9005A	CD9005B	CDA201E
CE2107I	CE2117A	CE2117B	CE2119B	CE2205B	CE2405A
CE3111C	CE3116A	CE3118A	CE3411B	CE3412B	CE3607B
CE3607C	CE3607D	CE3812A	CE3814A	CE3902B	

#### 2.2 INAPPLICABLE TESTS

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

## IMPLEMENTATION DEPENDENCIES

The following 201 tests have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

C24113H..K (4 tests) contain lines exceeding the implementation limit of 120 characters per line.

The following 20 tests check for the predefined type `LONG_INTEGER`; for this implementation, there is no such type:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45613C	C45614C	C45611C	C45632C	B52004D
C55B07A	B55B09C	B86001W	C86006C	CD7101F

C35404D, C45231D, B86001X, C86006E, and CD7101G check for a predefined integer type with a name other than `INTEGER`, `LONG_INTEGER`, or `SHORT_INTEGER`; for this implementation, there is no such type.

C35713B, C45423B, B86001T, and C86006H check for the predefined type `SHORT_FLOAT`; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than `FLOAT`, `LONG_FLOAT`, or `SHORT_FLOAT`; for this implementation, there is no such type.

A35801E checks that `FLOAT'FIRST..FLOAT'LAST` may be used as a range constraint in a floating-point type declaration; for this implementation, that range exceeds the range of safe numbers of the largest predefined floating-point type and must be rejected. (See section 2.3.)

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.

C45536A, C46013B, C46031B, C46033B, and C46034B contain length clauses that specify values for `'SMALL` that are not powers of two or ten; this implementation does not support such values for `'SMALL`.

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

IMPLEMENTATION DEPENDENCIES

D64005F..G (2 tests) use 10 levels of recursive procedure calls nesting; this level of nesting for procedure calls exceeds the capacity of the compiler.

B86001Y uses the name of a predefined fixed-point type other than type DURATION; for this implementation, there is no such type.

B91001H checks that an address clause for a task entry must not precede any entry; this implementation does not support interrupts. (See section 2.3.)

LA3004A..B, EA3004C..D, and CA3004E..F (6 tests) check pragma INLINE for procedures and functions; this implementation does not support pragma INLINE.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation 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 implementation does not support decimal 'SMALLs. (See section 2.3.)

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

AE2101C and EE2201D..E (2 tests) use instantiations of package SEQUENTIAL\_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT\_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

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

## IMPLEMENTATION DEPENDENCIES

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

The following 16 tests check operations on sequential, direct, and 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.

CE2107B..E	CE2107G..H	CE2107L	CE2110B	CE2110D
CE2111D	CE2111H	CE3111B	CE3111D..E	CE3114B
CE3115A				

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.

## 2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 1233 tests.

The following 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.

B44004D	B59001E	B73004B	BA1001A	BC3017A
---------	---------	---------	---------	---------

IMPLEMENTATION DEPENDENCIES

C34009D and C34009J were graded passed by Evaluation Modification as directed by the AVO. These tests check that 'SIZE for a composite type is greater than or equal to the sum of its components' 'SIZE values; but this issue is addressed by AI-00825, which has not been considered; there is not an obvious interpretation. This implementation represents array components whose length depends on a discriminant with a default value by implicit pointers into the heap space; thus, the 'SIZE of such a record type might be less than the sum of its components' 'SIZES, since the size of the heap space that is used by the varying-length array components is not counted as part of the 'SIZE of the record type. These tests were graded passed given that the Report.Result output was "FAILED" and the only Report.Failed output was "INCORRECT 'BASE'SIZE", from line 195 of C34009D and line 193 in C34009J.

A35801E was graded inapplicable by Evaluation Modification as directed by the AVO. The compiler rejects the use of the range FLOAT'FIRST..FLOAT'LAST as the range constraint of a floating-point type declaration because the bounds lie outside of the range of safe numbers (cf. LRM 3.5.7:12).

C83030C and C86007A were graded passed by Test Modification as directed by the AVO. These tests were modified by inserting "PRAGMA ELABORATE (REPORT);" before the package declarations at lines 13 and 11, respectively. Without the pragma, the packages may be elaborated prior to package Report's body, and thus the packages' calls to function REPORT.IDENT INT at lines 14 and 13, respectively, will raise PROGRAM\_ERROR.

B91001H was graded inapplicable by Evaluation Modification as directed by the AVO. This test checks that an address clause for an entry cannot precede that or any other entry of the task. This implementation does not support interrupts, and so rejects any address clause for an entry, regardless of placement.

The tests below were graded passed by Test Modification as directed by the AVO. These tests all use one of the generic support procedures, Length Check or Enum Check (in support files LENCHECK.ADA & ENUMCHEK.ADA), which use the generic procedure Unchecked Conversion. This implementation rejects instantiations of Unchecked Conversion with array types that have non-static index ranges. The AVO ruled that since this issue was not addressed by AI-00590, which addresses required support for Unchecked Conversion, and since AI-00590 is considered not binding under ACVC 1.1I, the support procedures could be modified to remove the use of Unchecked Conversion. Lines 40..43, 50, and 56..58 in LENCHECK and lines 42, 43, and 58..63 in ENUMCHEK were commented out.

CD1009A	CD1009I	CD1009M	CD1009V	CD1009W	CD1C03A
CD1C04D	CD2A21A..C	CD2A22J	CD2A23A..B	CD2A24A	CD2A31A..C
*CD2A81A	CD3014C	CD3014F	CD3015C	CD3015E..F	CD3015H
CD3015K	CD3022A	CD4061A			

IMPLEMENTATION DEPENDENCIES

\*CD2A81A, CD2A81B, CD2A81E, CD2A83A, CD2A83B, CD2A83C, and CD2A83E were graded passed by Test Modification as directed by the AVO. These tests check that operations of an access type are not affected if a 'SIZE clause is given for the type; but the standard customization of the ACVC allows only a single size for access types. This implementation uses a larger size for access types whose designated object is of type STRING. The tests were modified by incrementing the specified size \$ACC\_SIZE with '+ 64'.

Many of the Class A and Class C (executable) test files were combined into single procedures ("bundles") by the AVF, according to information supplied by the customer and guidance from the AVO. This bundling was done in order to reduce the processing time—compiling, linking, and downloading to the target. For each test that was bundled, its context clauses for packages Report and (if present) SYSTEM were commented out, and the modified test was inserted into the declarative part of a block statement in the bundle. The general structure of each bundle was:

```

WITH REPORT, TEXT_IO, SYSTEM;
PROCEDURE <BUNDLE_NAME> IS

-- repeated for each test

DECLARE
    <TEST FILE>    [a modified test is inserted here, ...]
BEGIN
    <TEST NAME>;  [... and invoked here]

EXCEPTION        --test is not expected to reach this exception handler
    WHEN OTHERS => REPORT.FAILED("unhandled exception ");
                    REPORT.RESULT;

END;
TEXT_IO.NEW_LINE;

--          [... repeated for each test in the bundle]

TEXT_IO.PUT_LINE ("GROUP TEST <BUNDLE_NAME> COMPLETED");
END <BUNDLE_NAME>;

```

The 1189 tests that were processed in bundles are listed below; each bundle is delimited by '<' and '>'.

<A21001A	A22002A	A22006B	A26004A	A26007A	A27003A	A27004A
A29002A	A29002B	A29002C	A29002D	A29002E	A29002F	A29002G
A29002H	A29002I	A29002J	A29003A	A2A031A>	<A32203B	A32203C
A32203D	A33003A	A34017C	A35101B	A35402A	A35502Q	A35502R
A35710A	A35801A	A35801B	A35801F	A35902C	A38106D	A38106E
A38199A	A39005B	A39005C	A39005D	A39005E	A39005F>	<A39005G
A54B01A	A54B02A	A55B12A	A55B13A	A55B14A	A62006D	A71002A
A71004A	A72001A	A73001I	A73001J	A74105B	A74106A	A74106B
A74106C	A74205E	A74205F>	<A83009A	A83009B	A83041B	A83041C
A83041D	A83A02A	A83A02B	A83A06A	A83A08A	A83C01C	A83C01D

## IMPLEMENTATION DEPENDENCIES

A83C01E	A83C01F	A83C01G	A83C01H	A83C01I	A83C01J	A85007D
A85013B	A87B59A>	<AB7006A	AC1015B	AC3106A	AC3206A	AC3207A>
<AD1A01A	AD1A01B	AD1D01E	AD7001B	AD7005A	AD7101A	AD7101C
AD7102A	AD7103A	AD7103C>	<AD7104A	AD7203B	AD7205B>	<C23001A
C23003A	C23006A	C24002A	C24002B	C24002C	C24003A	C24003B
C24003C	C24106A	C24113A	C24113B	C24113C	C24113D	C24113E>
<C24201A	C24202A	C24202B	C24202C	C24203A	C24203B	C24207A
C24211A	C25001A	C25001B	C25003A	C25004A	C26002B	C26006A>
<C26008A	C27001A	C2A001A	C2A001B	C2A001C	C2A002A	C2A006A
C2A008A	C2A009A	C2A021B>	<C32107A	C32107C	C32108A	C32108B
C32111A	C32111B>	<C32112A	C32112B	C32113A>	<C32114A	C32115A
C32115B>	<C32117A	C34001A	C34001C	C34001D	C34001F	C34002A
C34002C	C34003A	C34003C>	<C34004A	C34004C	C34005A	C34005C>
<C34005D	C34005F	C34005G	C34005I>	<C34005J	C34005L	C34005M
C34005O>	<C34005P	C34005R	C34005S	C34005U	C34006A	C34006F
C34006G	C34006J>	<C34006L	C34007A	C34007D	C34007F	C34007G>
<C34007I	C34007J	C34007M	C34007P>	<C34007R	C34007S>	<C34009A
C34009F	C34009G	C34009L	C34011B	C34012A	C34014A	C34014C>
<C34014E	C34014G	C34014H	C34014J	C34014L	C34014N	C34014P
C34014R	C34014T>	<C35503A	C35503B	C35503C	C35503D	C35503E
C35503F	C35503G	C35503H	C35503K>	<C35503L	C35503O	C35503P
C35504A	C35504B	C35505A	C35505B	C35505C>	<C35505D	C35505E
C35505F	C35507A	C35507B>	<C35507C	C35507E	C35507G	C35507H
C35507I	C35507J>	<C35507K	C35507L>	<C35706A	C35706B	C35706C
C35706D	C35706E>	<C35707A	C35707B	C35707C	C35707D	C35707E
C35708A	C35708B	C35708C	C35708D	C35708E>	<C35711A	C35711B
C35712A	C35712B	C35712C	C35713A	C35713C>	<C35801D	C35802A
C35802B	C35802C	C35802D	C35802E>	<C35902A	C35902B	C35902D
C35904A	C35904B	C35A02A	C35A03A	C35A03B	C35A03C	C35A03D>
<C35A03N	C35A03O	C35A03P>	<C35A03Q	C35A04A	C35A04B	C35A04C>
<C35A04D	C35A04N>	<C35A04O	C35A04P>	<C35A04Q	C35A05A	C35A05D
C35A05N>	<C35A05Q	C35A06A	C35A06B>	<C35A06D	C35A06N	C35A06O>
<C35A06P	C35A06Q	C35A06R	C35A06S	C35A07A	C35A07B	C35A07C>
<C35A07D	C35A07N	C35A07O	C35A07P	C35A07Q	C35A08B	C36003A>
<C36174A	C36180A	C36202A	C36202B	C36202C	C36203A	C36204A
C36204B	C36204C>	<C36205A	C36205B	C36205C	C36205D	C36205E
C36205F	C36205G	C36205H>	<C36205I	C36205J	C36205K	C36301A
C36301B	C36302A	C36303A	C36304A	C36305A>	<C37002A	C37003A
C37003B	C37005A	C37006A	C37007A	C37008A	C37008B>	<C37008C
C37009A	C37010A	C37010B	C37012A	C37102B	C37103A	C37105A
C37107A	C37108B	C37206A	C37207A	C37208A	C37208B	C37209A
C37209B	C37210A>	<C37211A	C37211B	C37211C	C37211D	C37211E
C37213A	C37213B	C37213C	C37213D>	<C37213E	C37213F	C37213G
C37213H>	<C37213J	C37213K	C37213L	C37214A>	<C37215A	C37215B>
<C37215C	C37215D	C37215E	C37215F	C37215G	C37215H	C37216A
C37217A	C37217B	C37217C>	<C37304A	C37305A	C37306A	C37307A
C37309A	C37310A	C37312A	C37402A	C37403A>	<C37404A	C37404B
C37405A	C37409A	C37411A	C38002A	C38002B	C38004A	C38004B
C38005A	C38005B	C38005C	C38006A	C38102A	C38102B	C38102C
C38102D	C38102E	C38104A	C38107A	C38107B>	<C38108A	C38201A
C38202A	C39006A	C39006B	C39006D	C39006E	C39006G	C39007A
C39007B	C39008A	C39008B	C39008C>	<C41101D	C41103A	C41103B
C41104A	C41105A	C41106A	C41107A	C41108A	C41201D	C41203A
C41203B>	<C41204A	C41205A	C41206A	C41207A	C41301A	C41303A

IMPLEMENTATION DEPENDENCIES

C41303B	C41303C	C41303E	C41303F	C41303G	C41303I	C41303J
C41303K	C41303M	C41303N	C41303O	C41303Q	C41303R	C41303S
C41303U	C41303V	C41303W	C41304A	<C41304B	C41306A	C41306B
C41306C	C41307A	C41307C	C41307D	C41308A	C41308C	C41308D
C41309A	<C41320A	C41321A	C41322A	C41323A	C41324A	C41325A
C41326A	C41327A	C41328A	<C41401A	C41402A	C41403A	C41404A
C42005A	C42006A	C42007A	C42007B	<C42007C	C42007D	C42007E
C42007F	C42007G	C42007H	C42007I	<C42007J	C42007K	C43003A
C43004B	C43004C	C43103A	C43103B	C43104A	<C43105A	C43105B
C43106A	C43107A	C43108A	C43204A	C43204C	C43204E	<C43204F
<C43204G	C43204H	C43204I	C43205A	C43205B	C43205C	C43205D
C43205E	C43205F	C43205G	C43205H	C43205I	C43205J	C43205K
C43206A	C43207A	C43207B	C43207C	<C43207D	C43208A	C43208B
C43209A	C43210A	C43211A	C43212A	C43212C	C43213A	<C43214A
C43214B	C43214C	C43214D	C43214E	C43214F	C43215A	C43215B
C43222A	<C43224A	C44003A	C44003D	C44003E	C44003F	C44003G
C45101A	C45101B	C45101C	C45101E	C45101G	C45101H	C45101I
C45101K	C45104A	C45111A	C45111B	C45111C	<C45111D	C45111E
C45112A	C45112B	C45113A	<C45114B	C45122A	C45122B	C45122C
C45122D	C45123A	C45123B	C45123C	<C45232A	C45232B	C45241A
C45241B	C45241C	C45241D	C45241E	<C45242A	C45242B	C45251A
C45252A	C45252B	C45253A	C45262A	<C45321A	C45321B	C45321C
C45321D	C45321E	<C45323A	C45331A	C45331D	C45332A	C45342A
C45343A	C45344A	C45345A	C45345B	C45345C	C45345D	<C45413A
C45421A	C45421B	C45421C	C45421D	C45421E	<C45505A	C45521A
C45521B	C45521C	C45521D	C45521E	<C45523A	C45524A	C45524B
C45524C	C45524D	C45524E	<C45532A	C45532B	C45532C	C45532D
C45532E	C45532F	C45532G	C45532H	C45532I	C45532J	C45532K
C45532L	<C45652A	C45662A	C45662B	C45672A	C46011A	C46012A
C46012B	C46012C	<C46012D	C46012E	<C46013A	C46014A	C46021A
C46023A	C46024A	C46031A	C46032A	C46033A	<C46041A	C46042A
C46043A	C46043B	<C46044A	C46044B	C46051A	C46051B	C46051C
<C46052A	C46053A	C46054A	C47002A	C47002B	C47002C	C47002D
C47003A	C47004A	C47005A	C47006A	C47007A	<C47008A	C47009A
C47009B	C48004A	C48004B	C48004C	C48004D	C48004E	C48004F
C48005A	C48005B	C48005C	C48006A	C48006B	<C48007A	C48007B
C48007C	C48008A	C48008B	C48008C	C48008D	C48009A	C48009B
C48009C	C48009D	C48009E	C48009F	C48009G	<C48009H	C48009I
C48009J	C48010A	C48011A	C48012A	C49020A	C49021A	C49022A
C49022B	C49022C	C49023A	C49024A	C49025A	C49026A	<C4A005A
C4A005B	C4A006A	C4A007A	C4A010A	C4A010B	C4A010D	C4A011A
C4A012A	C4A012B	C4A013A	C4A013B	C4A014A	<C51002A	C51004A
C52001A	C52001B	C52001C	C52005A	C52005B	C52005C	C52005D
C52005E	C52005F	<C52007A	C52008A	C52008B	C52009A	C52009B
C52010A	C52011A	C52011B	C52012A	C52012B	C52013A	<C52103B
C52103C	C52103F	C52103G	C52103H	C52103K	C52103L	<C52103M
C52103P	C52103Q	C52103R	C52103S	C52103X	C52104A	C52104B
C52104C	C52104F	<C52104G	C52104H	C52104K	C52104L	C52104M
C52104P	C52104Q	C52104R	C52104X	C52104Y	<C53004B	C53005A
C53005B	C53006A	C53006B	C53007A	C53008A	C54A03A	C54A04A
C54A06A	C54A07A	C54A11A	C54A13A	C54A13B	C54A13C	<C55B08A
C55B09A	C55B10A	C55B11A	C55B11B	C55B15A	C55B16A	C55C01A
C55C02A	C55C02B	C55C03A	C55C03B	C55D01A	C56002A	C57002A
C57003A	C57004A	C57004B	C57004C	C57005A	<C58004A	C58004B

IMPLEMENTATION DEPENDENCIES

C58004C	C58004D	C58004F	C58004G	C58005A	C58005B	C58005H
C58006A	C58006B	C59001B	C59002A	C59002B	C59002C	<C61008A
C61009A	C61010A	C62002A	C62003A	C62003B	C62004A	C62006A
C62009A	C63004A	C64002B	<C64004G	C64005A	C64005B	C64005C
C64103A	C64103B	C64103C	C64103D	C64103E	C64103F	<C64104A
C64104B	C64104C	C64104D	C64104E	C64104F	C64104G	C64104H
C64104I	C64104J	C64104K	C64104L	C64104M	C64104N	C64104O
C64105A	C64105B	C64105C	C64105D	C64105E	C64105F	<C64106A
C64106B	C64106C	C64106D	C64107A	C64108A	C64109A	C64109B
C64109C	C64109D	C64109E	<C64109F	C64109G	C64109H	C64109I
C64109J	C64109K	C64109L	<C64201B	C64201C	C64202A	C65003A
<C65003B	C65004A	C66002A	C66002C	C66002D	C66002E	C66002F
C66002G	C67002A	C67002B	C67002C	C67002D	C67002E	<C67003A
C67003B	C67003C	C67003D	C67003E	C67005A	C67005B	C67005C
C67005D	<C72001B	C72002A	C73002A	C73007A	C74004A	C74203A
C74206A	C74207B	C74208A	C74208B	C74209A	C74210A	C74211A
C74211B	C74302A	C74302B	C74305A	C74305B	C74306A	C74307A
<C74401D	C74401E	C74401K	C74401Q	C74402A	C74402B	C74406A
C74407B	C74409B	<C83007A	C83012D	C83022A	C83023A	C83024A
C83025A	<C83027A	C83027C	C83028A	C83029A	C83030A	<C83031A
C83031C	C83031E	C83032A	C83033A	C83051A	C83B02A	C83B02B
C83E02A	C83E02B	C83E03A	C83E04A	C83F01A	C83F03A	C84002A
C84005A	C84008A	C84009A	C85004B	C85005A	C85005B	C85005C
C85005D	<C85005E	C85005F	C85005G	C85006A	<C85006F	C85006G
<C87A05A	C87A05B	C87B02A	C87B02B	C87B03A	C87B04A	C87B04B
C87B04C	C87B05A	C87B06A	C87B07A	C87B07B	<C87B07C	C87B07D
C87B07E	C87B08A	C87B09A	C87B09B	C87B09C	C87B10A	C87B11A
C87B11B	C87B13A	C87B14A	C87B14B	C87B14C	C87B14D	<C87B15A
C87B16A	C87B17A	C87B18A	C87B18B	C87B19A	C87B23A	C87B24A
<C87B33A	C87B34A	C87B34B	C87B34C	C87B35A	C87B35B	C87B35C
C87B37A	C87B37B	C87B37C	C87B37D	C87B37E	C87B37F	C87B38A
C87B39A	<C87B40A	C87B41A	C87B42A	C87B43A	C87B44A	C87B45A
C87B45C	C87B47A	C87B48A	C87B48B	C87B50A	C87B54A	C87B57A
C87B62A	C87B62B	<CB1001A	CB1002A	CB1003A	CB1004A	CB1005A
CB1010A	CB1010B	CB1010C	CB1010D	<CB2004A	CB2005A	CB2006A
CB2007A	CB3003A	CB3003B	<CB3004A	CB4001A	CB4002A	CB4003A
CB4004A	CB4005A	CB4006A	CB4007A	CB4008A	CB4009A	CB4013A
CB5002A	CB7003A	CB7005A	<CC1004A	CC1005C	CC1010A	<CC1010B
CC1018A	CC1104C	CC1107B	CC1111A	CC1204A	CC1207B	CC1220A
CC1221A	CC1221B	CC1221C	CC1221D	<CC1222A	CC1224A	CC1225A
<CC1304A	CC1304B	CC1305B	CC1307A	CC1307B	CC1308A	CC1310A
<CC1311A	CC1311B	CC2002A	CC3004A	CC3007A	CC3011A	CC3011D
CC3012A	CC3015A	CC3106B	<CC3120A	CC3120B	CC3121A	CC3123A
CC3123B	CC3125A	CC3125B	CC3125C	CC3125D	<CC3126A	CC3127A
CC3128A	CC3203A	CC3207B	CC3208A	CC3208B	<CC3208C	CC3220A
CC3221A	CC3222A	CC3223A	CC3224A	CC3225A	<CC3230A	CC3231A
CC3232A	CC3233A	CC3234A	CC3235A	CC3236A	CC3240A	CC3305A
CC3305B	CC3305C	CC3305D	CC3406A	CC3406B	CC3406C	CC3406D
CC3407A	CC3407B	CC3407C	CC3407D	CC3407E	CC3407F	<CC3408A
CC3408B	CC3408C	CC3408D	CC3504A	CC3504B	CC3504C	CC3504D
CC3504E	CC3504F	<CC3504G	CC3504H	CC3504I	CC3504J	CC3504K
<CC3601A	CC3601C	<CC3603A	CC3606A	CC3606B	CC3607B	

IMPLEMENTATION DEPENDENCIES

\* This test listed in two explanations

CHAPTER 3  
PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report. For technical and sales information about this Ada implementation, contact:

Terry L. Dunbar  
TLD Systems, Ltd.  
3625 Del Amo Blvd.  
Suite 100  
Torrance, CA 90503

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

3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system — if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b and f, below).

## PROCESSING INFORMATION

a) Total Number of Applicable Tests	3760	
b) Total Number of Withdrawn Tests	95	
c) Processed Inapplicable Tests	114	
d) Non-Processed I/O Tests	0	
e) Non-Processed Floating-Point Precision Tests	201	
f) Total Number of Inapplicable Tests	315	(c+d+e)
g) Total Number of Tests for ACVC 1.11	4170	(a+b+f)

### 3.3 TEST EXECUTION

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The tests were grouped in bundles for more efficient processing. The contents of the magnetic tape were initially loaded on the Sun-4, and moved to the Data General MV/32 20000-2 using Ethernet.

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 and linked on the host computer system, as appropriate. The executable images were transferred to the target computer system by the communications link described above, and run. The results were captured on the host computer system.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B 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:

<u>Options   Switch</u>	<u>Effect</u>
No_Phase	Suppress displaying of phase times during compilation

All tests were executed with the Code Straightening, Global Optimizations, and automatic Inlining options enabled. Where optimizations are detected by the optimizer that represent deletion of test code resulting from unreachable paths, deleteable assignments, or relational tautologies or contradictions, such optimizations are reflected by informational or warning diagnostics in the compilation listings.

## PROCESSING INFORMATION

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

APPENDIX A  
MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. 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 here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
\$MAX_IN_LEN	120 -- Value of V
\$BIG_ID1	(1..V-1 => 'A', V => '1')
\$BIG_ID2	(1..V-1 => 'A', V => '2')
\$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
\$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
\$BIG_INT_LIT	(1..V-3 => '0') & "298"
\$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
\$BIG_STRING1	'"' & (1..V/2 => 'A') & "'"
\$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & "'"
\$BLANKS	(1..V-20 => ' ')
\$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
\$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"

MACRO PARAMETERS

\$MAX\_STRING\_LITERAL "' & (1..V-2 => 'A') & "'

The following table lists all of the other macro parameters and their respective values.

Macro Parameter	Macro Value
\$ACC_SIZE	32
\$ALIGNMENT	4
\$COUNT_LAST	2147483647
\$DEFAULT_MEM_SIZE	16#10000000#
\$DEFAULT_STOR_UNIT	16
\$DEFAULT_SYS_NAME	DGMV
\$DELTA_DOC	2.0**(-31)
\$ENTRY_ADDRESS	15
\$ENTRY_ADDRESS1	17
\$ENTRY_ADDRESS2	19
\$FIELD_LAST	127
\$FILE_TERMINATOR	ASCII.EDT
\$FIXED_NAME	NO_SUCH_FIXED_TYPE
\$FLOAT_NAME	NO_SUCH_FLOAT_TYPE
\$FORM_STRING	"
\$FORM_STRING2	"CANNOT RESTRICT FILE CAPACITY"
\$GREATER_THAN_DURATION	90000.0
\$GREATER_THAN_DURATION BASE LAST	I31073.0
\$GREATER_THAN_FLOAT_BASE LAST	2.0E+76
\$GREATER_THAN_FLOAT_SAFE LARGE	2.0E+75

MACRO PARAMETERS

\$GREATER\_THAN\_SHORT\_FLOAT\_SAFE\_LARGE  
 NO\_SUCH\_SHORT\_FLOAT\_TYPE  
  
 \$HIGH\_PRIORITY 200  
  
 \$ILLEGAL\_EXTERNAL\_FILE\_NAME1  
 BADFILENAME@.!  
  
 \$ILLEGAL\_EXTERNAL\_FILE\_NAME2  
 THISFILENAMEWOULDBEPERFECTLYLEGALIFITWERENOTSOLONG.SOTHERE  
  
 \$INAPPROPRIATE\_LINE\_LENGTH  
 -1  
  
 \$INAPPROPRIATE\_PAGE\_LENGTH  
 -1  
  
 \$INCLUDE\_PRAGMA1 PRAGMA INCLUDE ("A28006D1.TST")  
 \$INCLUDE\_PRAGMA2 PRAGMA INCLUDE ("B28006E1.TST")  
 \$INTEGER\_FIRST -2147483648  
 \$INTEGER\_LAST 2147483647  
 \$INTEGER\_LAST\_PLUS\_1 2147483648  
 \$INTERFACE\_LANGUAGE ASSEMBLY  
 \$LESS\_THAN\_DURATION -90000.0  
 \$LESS\_THAN\_DURATION\_BASE\_FIRST  
 -131073.0  
 \$LINE\_TERMINATOR ASCII.CR  
 \$LOW\_PRIORITY 1  
 \$MACHINE\_CODE\_STATEMENT  
 AC2'(OP=>1,ACS=>0,ACD=>2,OP2=>WMOV);  
 \$MACHINE\_CODE\_TYPE ACCUMULATOR  
 \$MANTISSA\_DOC 31  
 \$MAX\_DIGITS 15  
 \$MAX\_INT 2147483647  
 \$MAX\_INT\_PLUS\_1 2147483648  
 \$MIN\_INT -2147483648  
 \$NAME NO\_SUCH\_INTEGER\_TYPE

MACRO PARAMETERS

\$NAME_LIST	Af1750, DgMv
\$NAME_SPECIFICATION1	X2120A
\$NAME_SPECIFICATION2	X2120B
\$NAME_SPECIFICATION3	X3119A
\$NEG_BASED_INT	16#FFFFFFFFE#
\$NEW_MEM_SIZE	16#10000000#
\$NEW_STOR_UNIT	16
\$NEW_SYS_NAME	AF1750
\$PAGE_TERMINATOR	ASCII.FF
\$RECORD_DEFINITION	type Nullary is record Opcode : Nullary_Op; end record;
\$RECORD_NAME	NULLARY
\$TASK_SIZE	32
\$TASK_STORAGE_SIZE	2000
\$TICK	0.001
\$VARIABLE_ADDRESS	8#16000000200#
\$VARIABLE_ADDRESS1	8#16000000220#
\$VARIABLE_ADDRESS2	8#16000000240#
\$YOUR_PRAGMA	pragma export

APPENDIX B  
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.

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.

To delete default options:

TLD\_DEFAULTS DELETE

---

### 3.4 COMPILER OPTION SWITCHES

Compiler option switches provide control over various processing and output features of the compiler. These features include several varieties of listing output, the level and kinds of optimizations desired, the choice of target computer, and the operation of the compiler in a syntax checking mode only.

Keywords are used for selecting various compiler options. The complement keyword, if it exists, is used to disable a compiler option and is formed by prefixing the switch keyword with "NO\_".

Switches may be abbreviated to the number of characters required to uniquely identify the switch. For example, the switch "CROSSREF" (explained in the list below) may be uniquely identified by the abbreviation "CR" or any longer abbreviation. In the list of switches, on the following pages, the abbreviations are in bold, the optional extra characters are not bolded.

If an option is not specified by the user, a default setting is assumed. All specified compiler options apply to a single invocation of the compiler.

The default setting of a switch and its meaning are defined in the table below. The meaning of the complement form of a switch is normally the negation of the switch. For some switches, the complement meaning is not obvious; these complement switch keywords are listed separately.

In the description of the switches, the target dependent name *target* is used. The value of this symbol is determined by the value of the TARGET switch.

Compiler generated file specifications generally conform to host conventions. Thus, any generated filename is the source filename appended with the default-file extension. If the user specifies a full output filename, no default extension is provided.

## SWITCH NAME

## MEANING

## CALL TREE

**NO\_CALL\_TREE** -- default

This switch is used in conjunction with /ELABORATE and /LIST to cause all .CTI files (corresponding to the complete set of object files being linked for this program) to be read in and a closure of all calls in the program to be computed. The results of this analysis is formatted into a subprogram call tree report.

Note: The call tree will be incomplete if any required compilation unit's .CTI files are missing.

**CHECKS** -- default

**CHECKS**(-check\_identifier(...))

**NO\_CHECKS**(-check\_identifier(...))

When the CHECKS switch is used, one or more check\_identifiers are specified and the specified run time checks are enabled. The status of run time checks associated with unmentioned check\_identifiers is unchanged.

Without any check\_identifiers, the NO\_CHECKS switch omits all run time checks. If one or more check\_identifiers are specified, the specified run time checks are omitted. The status of run time checks associated with unmentioned check\_identifiers is unchanged.

Checks can be eliminated selectively or completely by source statement pragma Suppress. Pragma Suppress overrides the checks switch.

Check\_identifiers are listed below and are described in the LRM, Section 11.7.

ACCESS_CHECK	DISCRIMINANT_CHECK	DIVISION_CHECK
ELABORATION_CHECK	INDEX_CHECK	LENGTH_CHECK
OVERFLOW_CHECK	RANGE_CHECK	STORAGE_CHECK

ADE Related Switch: /\$SUPPRESS

CONFIG=*value*  
 NOCONFIG=*value*

The CONFIG switch provides a conditional compilation (configuration) capability by determining whether or not source text, introduced or bracketed by special comment constructs, is compiled. For a single line:

```
--/value source-text
```

where source-text is compiled only if config - value is specified.

For multiple lines:

```
--/value line-1
--/value line-2
--/value line-3
.
.
.
--/value line n
```

where the construct

```
--(value line-1
line-2
line-3
.
.
.
--)value line-n
```

is equivalent.

NOTE: The braces (( and )) must appear in the source code; in this instance, they are not metacharacters. All of the text between --(value and --)value is compiled or skipped, depending upon whether or not the /config-value is present.

## CPL

This switch sets the number of characters per line (80 to 132) in the listing file.

**CROSSREF**  
**NO\_CROSSREF** -- default

This switch generates a cross reference listing that contains names referenced in the source code. The cross reference listing is included in the listing file; therefore, the LIST switch must be selected or CROSSREF has no effect.

ADE Related Command: PRETTY/CREF

**CTI (-CTI-file-spec)**  
**NO\_CTI** -- default

This switch generates a CASE tools interface file. The default filename is derived from the object filename, with a .CTI extension. This switch is required to support the Stack Analysis and/or Call Tree switches.

**DEBUG** -- default  
**NO\_DEBUG**

This switch selects the production of symbolic debug tables in the relocatable object file.

Alternate abbreviation: **DBG**, **NO\_DBG**  
ADE Related Switch: /DEBUG, /TABLE

**DELASSIGN** -- default  
**NO\_DELASSIGN**

This switch optimizes code by deleting redundant assignments.

NOTE: Use of this switch can cause erroneous source programs to execute with unexpected results if references to access objects are made without regard to the interference semantics of Ada.

**ELABORATOR**

This switch selects generation of a setup program that elaborates all compilation units on which the named subprogram depends and then calls the named program. This program will be the main program at link time.

**EXCEPTION\_INFO****NO\_EXCEPTION\_INFO** -- default

This switch generates a string in the relocatable object code that is the full pathname of the file being compiled. It generates extra instructions to identify the location at which an unhandled exception occurred. The **NO\_EXCEPTION\_INFO** switch suppresses the generation of the string and extra instructions. The source statement `pragma Suppress (ALL_CHECKS)` or `pragma Suppress (EXCEPTION_INFORMATION)` suppresses only the extra instructions.

**FULL\_CALL\_TREE** -- default

When the **FULL\_CALL\_TREE** switch is used, the compiler listing includes representations of every call. |

**INDENT-n****INDENT-3** -- default**NO\_INDENT**

This switch produces a formatted (indented) source listing. This switch assigns a value to the number of columns used in indentation; the value n can range from 0 to 15.

**INFO** -- default**NO\_INFO**

This switch produces all diagnostic messages. It suppresses the production of information-level diagnostic messages.

**INLINE** -- default**NO\_INLINE**

This default switch automatically inlines any procedure that is called only once. It may be disabled by adding the `noinline` option to the command line. Inlining is only implemented for calls that are made within the same compilation unit as the body of the procedure to be expanded. |

**INTSL****NO\_INTSL** -- default

This switch intersperses lines of source code with the assembly code generated in the macro listing. This switch is valid only if the **LIST**, **SOURCE**, and **MACRO** switches are selected. |

ADE Related Switch: `/ASSEMBLY` |

**INVERTED\_CALLS**

This switch is used to determine which calls led to the present one. A reversed order call tree is generated. ?  
?

**LIBRARY**-Ada-program-library-file-spec  
**LIBRARY**-target.LIB -- default

The **LIBRARY** switch specifies the file to be used for Ada Program Library. The default value of **target** in the Ada Program Library file spec is derived from the **TARGET** switch.  
ADE Related Switch: /**LIBRARY**

**LIST**(-listing-file-spec)  
**NO\_LIST** -- default

The **LIST** switch generates a listing file. The default file extension is **LST**. The **listing-file-spec** can be optionally specified.

**LOG**  
**NO\_LOG** -- default

This switch requests the compiler to write a compiler log, including command line options and the file spec of the Ada source file being compiled, to @**OUTPUT**.

**LPP**-60 -- default

The **LPP** switch assigns a value to the number of lines per page for listing. The value can range from 10 to 99.

**MACRO**  
**NO\_MACRO** -- default

The **MACRO** switch produces an assembly like object code listing appended to the source listing file. The **LIST** switch must be enabled for this switch to work.

**MAIN\_ELAB**  
**NOMAIN\_ELAB** -- default

The **MAIN\_ELAB** switch is used to inform the compiler that the compilation unit being compiled is to be treated as a user-defined elaboration, or setup, program.

Note: The **XTRA** switch is required when **MAIN\_ELAB** is to be used.

**MAKELIB**(-parent-APL-spec)  
**NO\_MAKELIB** -- default

The **MAKELIB** switch creates a new Ada Program Library (APL) file. **MAKELIB** should be used with caution because it creates a new APL file in the default directory even if another APL file of the same name existed.

The new APL file is created in the current working directory with the name **target.LIB** unless the **LIBRARY** switch is used.

If **MAKELIB** is used without a parent, a new library is created with the default RTS specification. This specification is derived from the name **TLD\_LIB\_target**. See the target dependent compiler sections for further explanations of this name.

ADE Related Commands: **LIBCREATE**, **LIBSEARCHLIST**

**MAXERRORS**=n  
**MAXERRORS**=500 -- default

This switch assigns a value limit to the number of errors forcing job termination. Once this value is exceeded, the compilation is terminated. Information-level diagnostic messages are not included in the count of errors forcing termination. The specified value's range is from 0 to 500.

**MODEL**=MATH -- MV target  
**MODEL**=MATH -- MV target

By default, the compiler produces code for the generic or standard target. The model switch allows the user to specify a nonstandard model for the target.

For the **MV-HAWK** target, the **MODEL** switch may be set to **MATH** to indicate that code generation should take advantage of the intrinsic math instructions.

**NEG\_OFFSET**

The **NEG\_OFFSET** switch is used to get around a deficiency in the MV linker. Negative offsets can cause a "relocation overflow" error if this option is not used.

**OBJECT**=(object-file-spec)  
**OBJECT** -- default  
**NO\_OBJECT**

The **OBJECT** switch produces a relocatable object file. The default file extension is ".OBJ".

**OPT** -- default  
**NO\_OPT**

The OPT switch enables global optimization of the compiled code.

**PARM**  
**NO\_PARM** -- default

The PARM switch causes all option switches governing the compilation, including the defaulted option switches, to be included in the listing file. The LIST option must also be selected or PARM will have no effect. User specified switches are preceded in the listing file by a leading asterisk (\*).

**PHASE** -- default  
**NO\_PHASE**

The NO\_PHASE switch suppresses the display of phase names during compilation.

#### **REF\_ID\_Case-option**

The Ref\_Id\_Case switch is used to determine how variable names appear in the compiler listing. The options for this switch are:

All_Lower	--	All variable names are in lower case.
All_Underlined	--	All variable names are underlined.
All_Upper	--	All variable names are in upper case.
As_Is	--	All variable names appear as is.
Initial_Caps	--	All variable names have initial caps.
Insert_Underscore	--	All variable names have underscores inserted.

#### **REF\_KEY\_Case-option**

The Ref\_Key\_Case switch is used to determine how Ada key words appear in the compiler listing. The options for this switch are:

All_Lower	--	All Ada key words are in lower case.
All_Underlined	--	All Ada key words are underlined.
All_Upper	--	All Ada key words are in upper case.
As_Is	--	All Ada key words appear as is.
Initial_Caps	--	All Ada key words have initial caps.
Insert_Underscore	--	All Ada key words have underscores inserted.

REFORMAT(-reformat-file-spec)  
NO\_REFORMAT -- default

This switch causes TLDada to reformat the source listing in the listing file and, if a reformat-file-spec is present, to generate a reformatted source file. The default extension of the new source file is ".RFM".

ADE Related Command: PRETTY

SOURCE -- default  
NO\_SOURCE

This switch causes the input source program to be included in the listing file. Unless they are suppressed, diagnostic messages are always included in the listing file.

ADE Related Switch: /ERRORS

STACK ANALYSIS  
NO\_STACK\_ANALYSIS -- default

This switch is used with the ELABORATOR switch to cause all CTI files (corresponding to the complete set of object files being linked for this program) to be read in. The subprogram call tree is analyzed to compute stack requirements for the main program and each dependent task.

NOTE: The tree will be incomplete if any required compilation unit's CTI files are missing.

SYNTAX\_ONLY  
NO\_SYNTAX\_ONLY -- default

This switch performs syntax and semantic checking on the source program. No object file is produced and the MACRO switch is ignored. The Ada Program Library is not updated.

TARGET=MV -- default  
TARGET=HAWK

This switch selects the target computer for which code is to be generated for this compilation. "HAWK" selects the Rolm HAWK architecture running under AOS/VS or ARTS/32. "MV" selects the Eclipse MV architecture operating under AOS/VS, AOS/VS II, or RT/32.

**WARNINGS** -- default  
**NO\_WARNINGS**

The **WARNINGS** switch outputs warning and higher-level diagnostic messages.

The **NO\_WARNINGS** switch suppresses the output of both warning-level and information-level diagnostic messages.

**WRITE\_ELAB**  
**NO\_WRITE\_ELAB** -- default

The **WRITE\_ELAB** switch is used to obtain an Ada source file which represents the main elaboration "setup" program created by the compiler. The **MAIN\_ELAB** switch may not be used at the same time as the **ELAB** switch.

**XTRA**  
**NO\_XTRA** -- default

This switch is used to access features under development. See the description of this switch in Section 3.9.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics 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. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

package STANDARD is

.....

type INTEGER is range -2147483648 .. 2147483647;

type FLOAT is digits 6 range -2.26156E+74 .. 2.26156E+74;

type DURATION is delta 2.0\*\*(-12) range -86400.0 .. 86400.0;

type SHORT\_INTEGER is range -32768 .. 32767;

type LONG\_FLOAT is digits 15  
range -2.2615642491634E+74 .. 2.2615642491634E+74;

.....

end STANDARD;

## APPENDIX F OF THE Ada STANDARD

In the customer's Appendix F documentation that constitutes this appendix, some information appears to be inaccurate or incomplete; the AVF offered the customer an opportunity to redress these points, but the customer declined to do so.

The customer declined to provide the AVF with an updated list of all compiler/linker options and the options used specifically for this validation.

The customer states that  $DURATION'DELTA = 2.0^{*(-12)}$ . Although other implementations have a value of  $2.0^{*(-14)}$  and therefore should possibly be  $2.0^{*(-14)}$ . The customer declined to provide the AVF with an updated Appendix F for this implementation; therefore we cannot be certain as to the correct value.

The customer declined to provide an Appendix F containing a package SYSTEM specification.

On page C-4, the customer states several inaccuracies concerning length clauses being not supported for 'SIZE except when applied to task types and 'Storage Size when applied to access types. Also length clauses are supported for 'Small. The customer declined to provide the AVF with an updated Appendix F for this implementation.

The Ada language definition allows for certain machine dependencies in a controlled manner. No machine-dependent syntax of semantic extensions or restrictions are allowed. The only allowed implementation-dependencies correspond to implementation-dependent pragmas and attributes, certain machine-dependent conventions as mentioned in chapter 13 of the MIL-STD-1815A; and certain allowed restrictions on representation clauses.

The full definition of the implementation-dependent characteristics of the TLD MV/MV Ada Compiler System is presented in this Appendix F.

#### Implementation-Dependent Pragmas

The TLD ACS supports the following implementation dependent pragmas.

```
Pragma Export (Language_name, Ada_entity_name, {String});
```

This pragma is a complement to Pragma Interface and instructs the compiler to make the entity named available for reference by a foreign language module. The language name identifies the language in which the foreign module is coded. The only foreign language presently supported is Assembly. Ada and JOVTAL are permitted and presently mean the same as Assembly but the semantics of their use are subject to redefinition by future releases of the compiler. If the optional third parameter is present, the string provides the name by which the entity may be referenced by the foreign module. The contents of this string must conform to the conventions for the indicated foreign language and the linker being used. No check is made by the compiler to insure that these conventions are obeyed.

Only objects having static allocation and subprograms are supported by pragma Export. If the Ada entity named is a subprogram, this pragma must be placed within the declarative region of the named subprogram. If the name is that of an object, the pragma must be placed within the same declarative region and following the object declaration. It is the responsibility of the programmer to insure that the subprogram and object are elaborated before the reference is made.

```
pragma If (Compile_Time_Expression);
pragma Elsif (Compile_Time_Expression);
pragma Else;
pragma Endif;
```

These Source directives may be used to enclose conditionally compiled source to enhance program portability and configuration adaptation. These directives may occur at the place that language defined pragmas, statements, or declarations may occur. Source code following these pragmas will be compiled or ignored similar to the semantics of the corresponding Ada statements depending upon whether the Compile\_Time\_Expression is true

## APPENDIX F OF THE Ada STANDARD

or false, respectively. The primary difference between these pragmas and the corresponding Ada Statements are that the pragmas may enclose declarations and other pragmas.

### Implementation-Dependent Attributes

None.

### Representation Clause Restrictions

Pragma Pack is not supported.

Length clauses are not supported for 'SIZE except when applied to task types.

Length clauses are not supported for 'Storage\_Size when applied to access types.

Length clauses are not supported for 'Storage\_Size when applied to a task type and denote the number of words of stack to be allocated to the task.

Length clauses are not supported for 'Small.

Enumeration representation clauses are supported for value ranges of Integer'First to Integer'Last.

Record representation clauses are supported to arrange record components within a record. Record components may not be specified to cross a word boundary unless they are arranged to encompass two or more whole words. A record component of type record that has itself been "rep specificationed" may only be allocated at bit 0. Bits are numbered from left to right with bit 0 indicating the sign bit.

The alignment clause is not supported.

Address clauses are supported for variable objects and designate the virtual address of the object. The TLD Ada Compiler System treats the address specification as a means to access objects allocated by other than Ada means and accordingly does not treat the clause as a request to allocate the object at the indicated address.

Address clauses are not supported for constant objects, packages, tasks, or task entries.

### Implementation-Generated Names

The TLD Ada Compiler System defines no implementation dependent names for compiler generated components.

## Address Clause Expressions

Address expression values and type Address represent a location in logical memory (in the program's current address space). For objects, the address specifies a location within the logical address space. The 'Address attribute applied to a subprogram represents a 32 bit word address within the logical instruction space.

## Unchecked Conversion Restrictions

None.

## I/O Package Characteristics

The following implementation-defined types are declared in Text\_Io.

subtype Count is integer range 0 .. 2\_147\_483\_647;

subtype Field is integer range 0 .. 127;

## Other System Dependencies

LRM Chapter 1.

None.

LRM Chapter 2.

Maximum source line length — 120 characters.

Source line terminator — Determined by the Editor used.

Maximum name length — 120 characters.

External representation of name characters.

Maximum String literal — 120 characters.

LRM Chapter 3.

LRM defined pragmas are recognized and processed as follows:

Controlled — Has no effect.

Elaborate — As described in the LRM.

Inline — Not presently supported.

## APPENDIX F OF THE Ada STANDARD

Interface -- Supported as a means of importing foreign language components into the Ada Program Library. May be applied either to a subprogram declaration as being specially implemented, -- read Interface as Import --, or to an object that has been declared elsewhere. Interface languages supported are Assembly for calling assembly language routines; and MV 32 for defining built in instruction procedures. An optional third parameter is used to define a name other than the name of the Ada subprogram for interfacing with the linker.

List -- As defined in the LRM.

Memory Size -- Has no effect.

Optimize -- Has no effect. Optimization controlled by compiler command option.

Pack -- Has no effect.

Page -- As defined in the LRM.

Priority -- As defined in the LRM. Priority may range from 1 to 200. Default priority is 1.

Shared -- As defined in the LRM. May be applied to scalar objects only.

Storage Unit -- Has no effect.

Suppress -- As defined in the LRM for suppressing checks; all standard checks may be suppressed individually as well as "Exception Info" and "All Checks". Suppression of Exception Info eliminates data used to provide symbolic debug information in the event of an unhandled exception. The All Checks selection eliminates all checks with a single pragma. In addition to the pragma, the compiler permits control of check suppression by command line option without the necessity of source changes.

System Name -- Has no effect.

Number declarations are not assigned addresses and their names are not permitted as a prefix to the 'address attribute. (Clarification only).

Objects are allocated by the compiler to occupy one or more 16 bit words. Only in the presence record representation clauses are objects allocated to less than a word.

Except for access objects, uninitialized objects contain an undefined value. An attempt to reference the value of an uninitialized object is not detected.

The maximum number of enumeration literals of all types is limited only by

available symbol table space.

The predefined integer types are:

Short\_Integer range -32\_768 .. 32\_767 and is implemented as a 16 bit value.  
 Integer range -2\_147\_483\_648 .. 2\_147\_483\_647 and implemented a 32 bit value.  
 Long\_Integer is not supported.  
 System.Min\_Int is -2\_147\_483\_648.  
 System.Max\_Int is 2\_147\_483\_647.

The predefined real types are:

Float digits 6.  
 Long\_Float digits 15.  
 Short\_Float is not presently supported.  
 System.Max\_Digits is 15 digits.

Fixed point is implemented as 16-bit or 32-bit data as is appropriate for the range and delta.

Index constraints as well as other address values such as access types are limited to 29 bits of value.

The maximum array size is limited to the size of virtual address.

The maximum String length is the same as for other arrays.

Access objects are implemented as an unsigned 32 bit integer. The access literal Null is implemented as two words of zero.

There is no limit on the number of dimensions of an array type. Array types are passed as parameters opposite unconstrained formal parameters using a 3 word dope vector illustrated below:

Word address of first element
Low bound value of first dimension
Upper bound value of first dimension

Additional dimension bounds follow immediately for arrays with more than one dimension.

LRM Chapter 4.

Machine\_Overflows is True.

## APPENDIX F OF THE Ada STANDARD

Pragma Controlled has no effect since garbage collection is never performed.

### LRM Chapter 5.

The maximum number of statements in an Ada source program is undefined and limited only by the Symbol Table space.

Case statements unless they are quite sparse, are allowed as indexed jump vectors and are, therefore, quite fast.

Loop statements with a for implementation scheme are implemented most efficiently if the range is in reverse and down to zero.

Data declared in block statements is elaborated as part of its containing scope.

### LRM Chapter 6.

Arrays, records and task types are passed by reference.

Pragma Inline is not presently supported for subprograms.

### LRM Chapter 7.

Package elaboration is performed dynamically permitting a warm restart without the necessity to reload the program.

### LRM Chapter 8.

### LRM Chapter 9.

Task objects are implemented as access types pointing to a Task Information Block (TIB).

Type Time in package Calendar is declared as a record containing two double precision integer values: the date in days and the real time clock.

Pragma Priority is supported with a value of 1 to 200.

Pragma Shared is supported for scalar objects.

### LRM Chapter 10.

Multiple Ada Program Libraries are supported with each library containing an optional ancestor library. The predefined packages are contained in the TLD standard library, MV.LIB.

### LRM Chapter 11.

Exceptions are implemented by the TLD Ada Compiler System to take advantage of the normal policy in embedded computer system design to reserve 50% of

the duty cycle. By executing a small number of instructions in the prologue of a procedure or block containing an exception handler, a branch may be taken, at the occurrence of an exception, directly to a handler rather than performing the time consuming code of unwinding procedure calls and stack frames. The philosophy taken is that an exception signals an exceptional condition, perhaps a serious one involving recovery or reconfiguration, and that quick response in this situation is more important and worth the small throughput tradeoff in a real time environment.

#### LRM Chapter 12.

A single generic instance is generated for a generic body. Generic specifications and bodies need not be compiled together nor need a body be compiled prior to the compilation of an instantiation. Because of the single expansion, this implementation of generics tend to be more favorable of space savings. To achieve this tradeoff, the instantiations must by nature be more general and are, therefore, somewhat less efficient timewise.

#### LRM Chapter 13.

Representation clause support and restrictions are defined above.

A comprehensive Machine\_Code package is provided and supported.

Unchecked\_Deallocation and Unchecked\_Conversion are supported.

The implementation dependent attributes are all supported except 'Storage\_Size for an access type.

#### LRM Chapter 14.

Full file I/O operations are supported and interface to AOS/VS operations. Text\_Io and Low\_Level\_Io are supported.