Ada Compiler Validation Summary Report, Siemens AG, Siemens BS2000 Ada Compiler V2.0, Siemens 7.590G (host and target), 89030611.10059

IABG, Ottobrunn, Federal Republic of Germany.

IABG, Ottobrunn, Federal Republic of Germany.

Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

IABG, Ottobrunn, Federal Republic of Germany.

Approved for public release; distribution unlimited.


Siemens BS2000 Ada Compiler V2.0, Siemens AG, IABG, Ottobrunn, Siemens 7.590G under BS2000/V9.0 (host and target), ACVC 1.
Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890306I1.10059
Siemens AG
Siemens BSZ000 Ada Compiler V2.0
Siemens 7.5POG Host and Target

Completion of On-Site Testing:
6th March 1989

Prepared By:
IABG mbH, Abt S27
Einsteinstr 20
D-8012 Ottobrunn
West Germany

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Accession For
NTIS CR&AI
DTIC TAB
Unannounced
Justification

By
Distribution

Availability Codes
A  Open to Public
I  Available Internally
S  Special

Dist
A-1
Ada Compiler Validation Summary Report:

Compiler Name: Siemens BS2000 Ada Compiler V2.0
Certificate Number: 89030611.10059
Host and Target: Siemens 7.590G under BS2000/V9.0
Testing Completed 6th March 1989 Using ACVC 1.10

This report has been reviewed and is approved.

IABG mbH Abt SZI
Dr. S. Heilbrunner
Einsteinstr 20
D-8012 Ottobrunn
West Germany

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director
Washington D.C. 20301
Ada Compiler Validation Summary Report:

Compiler Name: Siemens BS2000 Ada Compiler V2.0
Certificate Number: 89030611.10059
Host and Target: Siemens 7.590G under BS2000/V9.0
Testing Completed 6th March 1989 Using ACVC 1.10

This report has been reviewed and is approved.

IABG mbH Abt SZ1
Dr. S. Heilbrunner
Einsteinstr 20
D-8012 Ottobrunn
West Germany

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomon
Director
Washington D.C. 20301
CONTENTS

CHAPTER 1 INTRODUCTION
1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-1
1.2 USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3 REFERENCES 1-3
1.4 DEFINITION OF TERMS 1-3
1.5 ACVC TEST CLASSES 1-4

CHAPTER 2 CONFIGURATION INFORMATION
2.1 CONFIGURATION TESTED 2-1
2.2 IMPLEMENTATION CHARACTERISTICS 2-2

CHAPTER 3 TEST INFORMATION
3.1 TEST RESULTS 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER 3-2
3.4 WITHDRAWN TESTS 3-2
3.5 INAPPLICABLE TESTS 3-2
3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS 3-6
3.7 ADDITIONAL TESTING INFORMATION 3-7
3.7.1 Prevalidation 3-7
3.7.2 Test Method 3-7
3.7.3 Test Site 3-8

APPENDIX A DECLARATION OF CONFORMANCE
APPENDIX B APPENDIX F OF THE Ada STANDARD
APPENDIX C TEST PARAMETERS
APPENDIX D WITHDRAWN TESTS
APPENDIX E COMPILER AND LINKER OPTIONS
CHAPTER 1
INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies—for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:
INTRODUCTION

. To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

. To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard

. To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 6th March 1989 at Siemens AG, Munich.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO 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 compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

IABG mbH, Abt SZT
Einsteinstr 20
D-8012 Ottobrunn
West Germany

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311
1.3 REFERENCES


1.4 DEFINITION OF TERMS

ACVC  The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary  An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.


Applicant  The agency requesting validation.

AVF  The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO  The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler  A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test  An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host  The computer on which the compiler resides.
1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada 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, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

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 every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.
INTRODUCTION

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters—for example, the number of identifiers permitted in a compilation or the number of units in a library—a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time—that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the Package REPORT and the procedure CHECKFILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It 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 procedure CHECKFILE 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 CHECKFILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be
customized according to implementation-specific values—for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.
2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Siemens BS2000 Ada Compiler V2.0
ACVC Version: 1.10
Certificate Number: 89030611.10059
Host and Target Computer:
  Machine: Siemens 7.5906
  Operating System: BS2000/V9.0
  Memory Size: 64 MB
CONFIGURATION INFORMATION

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests)).

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests)).

b. Predefined types.

(1) This implementation supports the additional predefined type SHORT_INTEGER in the package STANDARD. (See tests B86001T..Z (7 tests)).

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35905A.)
(4) CONSTRAINT_ERROR is raised for integer and largest integer and no exception is raised for smallest integer when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) CONSTRAINT_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is not gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z (26 tests).)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

(1) Test C36003A deals with an array type declaration with SYSTEM.MAX_INT components. The parameters of the type declaration can be evaluated at compile time. No type descriptor is provided by the compiler. The elaboration of the type declaration will only check the index bounds (and not the length of the array). No exception will be raised. (See test C36003A.)

(2) Test C36202A deals with an array type declaration with INTEGER'LAST + 2 components. The parameters of the type declaration cannot be evaluated at compile time. A type descriptor is provided by the compiler which contains the length of the array. The elaboration of the type descriptor will calculate the length of the array. NUMERIC_ERROR will be
CONFIGURATION INFORMATION

raised since the length exceeds the admissible range. (See test C36202A.)

(3) NUMERIC_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared, for the same reasons as for test C36202A above. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

(6) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(7) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

g. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

h. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)
(2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

i. Pragmas.

(1) The pragma INLINE is not supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

j. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

(3) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(4) Generic non-library package bodies as subunits can be compiled in separate compilations. (See test CA2009C.)

(5) Generic non-library subprogram bodies can be compiled in separate compilations from their stubs. (See test CA2009F.)

(6) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

(7) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

(8) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(9) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)
k. Input and output.

(1) The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)

(4) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)

(5) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)

(6) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102C and CE2102X.)

(7) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(8) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

(9) Overwriting to a sequential file truncates to the last element written. (See test CE220BB.)

(10) Temporary sequential files are given names and not deleted when closed. (See test CE2108A.)

(11) Temporary direct files are given names and not deleted when closed. (See test CE2108C.)

(12) Temporary text files are given names and not deleted when closed. (See test CE3112A.)

(13) More than one internal file can be associated with each external file for sequential files when reading only. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(14) More than one internal file can be associated with each external file for direct files when reading only. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)
(15) More than one internal file can be associated with each external file for text files when reading only. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)
CHAPTER 3
TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 43 tests had been withdrawn because of test errors. The AVF determined that 401 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 16 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

<table>
<thead>
<tr>
<th>RESULT</th>
<th>TEST CLASS</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Passed</td>
<td>127</td>
<td>1130</td>
</tr>
<tr>
<td>Inapplicable</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Withdrawn</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>TOTAL</td>
<td>130</td>
<td>1140</td>
</tr>
</tbody>
</table>
3.3 SUMMARY OF TEST RESULTS BY CHAPTER

<table>
<thead>
<tr>
<th>RESULT</th>
<th>CHAPTER 2</th>
<th>CHAPTER 3</th>
<th>CHAPTER 4</th>
<th>CHAPTER 5</th>
<th>CHAPTER 6</th>
<th>CHAPTER 7</th>
<th>CHAPTER 8</th>
<th>CHAPTER 9</th>
<th>CHAPTER 10</th>
<th>CHAPTER 11</th>
<th>CHAPTER 12</th>
<th>CHAPTER 13</th>
<th>CHAPTER 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passed</td>
<td>198</td>
<td>572</td>
<td>544</td>
<td>245</td>
<td>172</td>
<td>99</td>
<td>159</td>
<td>333</td>
<td>131</td>
<td>36</td>
<td>252</td>
<td>254</td>
<td>278</td>
</tr>
<tr>
<td>N/A</td>
<td>14</td>
<td>77</td>
<td>136</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>115</td>
<td>43</td>
</tr>
<tr>
<td>Wdrn</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>35</td>
<td>4</td>
</tr>
<tr>
<td>TOTAL</td>
<td>213</td>
<td>650</td>
<td>680</td>
<td>248</td>
<td>172</td>
<td>99</td>
<td>166</td>
<td>334</td>
<td>137</td>
<td>36</td>
<td>253</td>
<td>404</td>
<td>325</td>
</tr>
</tbody>
</table>

3.4 WITHDRAWN TESTS

The following 43 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

- E28005C
- A39005G
- B97102E
- BC3009B
- CD2A62D
- CD2A63A
- CD2A63B
- CD2A63C
- CD2A63D
- CD2A66A
- CD2A66B
- CD2A66C
- CD2A66D
- CD2A73A
- CD2A73B
- CD2A73C
- CD2A73D
- CD2A76A
- CD2A76B
- CD2A76C
- CD2A81G
- CD2A83G
- CD2A84N
- CD2A84M
- CD50110
- CD2B15C
- CD7205C
- CD2D11B
- CD5007B
- CD7105A
- CD7104B
- CD7004C
- ED7005D
- ED7006A
- CD7006D
- CD7105A
- CD7203B
- CD7204B
- CD7205D
- CE2107I
- CE3111C
- CE3301A
- CE3411B

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 401 tests were inapplicable for the reasons indicated:

a. The following 201 tests are inapplicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:
b. C35508I, C35508J, C35508M, and C35508N are inapplicable because they include enumeration representation clauses for BOOLEAN types in which the representation values are other than \((\text{FALSE} \Rightarrow 0, \text{TRUE} \Rightarrow 1)\). Under the terms of AI-00325, this implementation is not required to support such representation clauses.

c. C35702A and B86001T are inapplicable because this implementation supports no predefined type SHORT_FLOAT.

d. C35702B and B86001U are inapplicable because this implementation supports no predefined type LONG_FLOAT.

e. The following 16 tests are inapplicable because this implementation does not support a predefined type LONG_INTEGER:

\[
\begin{array}{cccc}
C45231C & C45304C & C45502C & C45503C \\
C45504F & C45611C & C45613C & C45614C \\
C45632C & B52004D & C55807A & B55809C \\
CD7101F & & & \\
\end{array}
\]

f. C45531M..P (4 tests) and C45532M..P (4 tests) are inapplicable because this implementation has a value of MAX_MANTISSA of less than 48.

g. B86001F is inapplicable because, for this implementation, the package TEXT_IO is dependent upon package SYSTEM. These tests recompile package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete.

h. B86001X, C45231D, and CD7101G are inapplicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

i. B86001Y is inapplicable because this implementation supports no predefined fixed-point type other than DURATION.

j. B86001Z is inapplicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

k. LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F are inapplicable because this implementation does not support pragma INLINE.
TEST INFORMATION

1. The following 14 tests are inapplicable because of restrictions on 'SIZE length clauses for floating point types:

   CD1009C  CD2A41A  CD2A41B  CD2A41E  CD2A42A
   CD2A42B  CD2A42C  CD2A42D  CD2A42E  CD2A42F
   CD2A42G  CD2A42H  CD2A42I  CD2A42J

m. The following 22 tests are inapplicable because of restrictions on 'SIZE length clauses as detailed in Appendix F of the Ada Standard:

   CD2A61A  CD2A61B  CD2A61F  CD2A61H  CD2A61I
   CD2A61J  CD2A62A  CD2A62B  CD2A71A  CD2A71B
   CD2A72A  CD2A72B  CD2A84B  CD2A84C  CD2A84D
   CD2A84E  CD2A84F  CD2A84G  CD2A84H  CD2A84I
   CD2A84K  CD2A84L

n. CDZB15B is inapplicable because a collection size larger than the size specified was allocated by this implementation.

o. The following 76 tests are inapplicable because this implementation does not support address clauses:

   CD5003B  CD5003C  CD5003D  CD5003E  CD5003F
   CD5003G  CD5003H  CD5003I  CD5011A  CD5011B
   CD5011C  CD5011D  CD5011E  CD5011F  CD5011G
   CD5011H  CD5011I  CD5011K  CD5011L  CD5011M
   CD5011N  CD5011Q  CD5011R  CD5011S  CD5012A
   CD5012B  CD5012C  CD5012D  CD5012E  CD5012F
   CD5012G  CD5012H  CD5012I  CD5012J  CD5012L
   CD5012M  CD5013A  CD5013B  CD5013C  CD5013D
   CD5013E  CD5013F  CD5013G  CD5013H  CD5013I
   CD5013K  CD5013L  CD5013M  CD5013N  CD5013O
   CD5013R  CD5013S  CD5014A  CD5014B  CD5014C
   CD5014D  CD5014E  CD5014F  CD5014G  CD5014H
   CD5014I  CD5014J  CD5014K  CD5014L  CD5014M
   CD5014N  CD5014O  CD5014R  CD5014S  CD5014T
   CD5014U  CD5014V  CD5014W  CD5014X  CD5014Y
   CD5014Z

p. AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_10 with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

q. AE2101H, EE2401D, and EE2401E use instantiations of package DIRECT_10 with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.
r. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.
s. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.
t. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.
u. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.
v. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.
w. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.
x. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.
y. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.
z. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

aa. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.
ab. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.
ac. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.
ad. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.
ae. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.
af. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

ag. CE2107B..E (4 tests), CE2107L, CE2110B and CE2111D are inapplicable because multiple internal files cannot be associated with the same external file when one or more files is writing for sequential files. The proper exception is raised when multiple access is attempted.
TEST INFORMATION

ah. CE2107G..H (2 tests), CE2110D, and CE2111H are inapplicable because multiple internal files cannot be associated with the same external file when one or more files is writing for direct files. The proper exception is raised when multiple access is attempted.

ai. CE3102E is inapplicable because text file CREATE with IN_FILE mode is supported by this implementation.

aj. CE3102F is inapplicable because text file RESET is supported by this implementation.

ak. CE3102G is inapplicable because text file deletion of an external file is supported by this implementation.

al. CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

am. CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.

an. CE3102K is inapplicable because text file OPEN with OUT_FILE mode is not supported by this implementation.

ao. CE3111B, CE3111D..E (2 tests), CE3114B, and CE3115A are inapplicable because multiple internal files cannot be associated with the same external file when one or more files is writing for text files. The proper exception is raised when multiple access is attempted.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 16 tests.

The following tests required the inclusion of

"PRAGMA ELABORATE (REPORT);"
The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A  B24007A  B24009A  B35302A  B38003A  B38009A
B38009B  B49003A  B49005A  B67001C  B95032A  B97103E

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the Siemens B52000 Ada Compiler V2.0 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the Siemens B52000 Ada Compiler V2.0 using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host and Target computer: Siemens 7.590G
Host and Target operating system: BS2000/V9.0

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the Siemens 7.590G. Results were printed from the host computer.

The compiler was tested using command scripts provided by Siemens AG and reviewed by the validation team. The compiler was tested using all default option settings. See Appendix E for the list of options and their meaning.
Tests were compiled, linked, and executed (as appropriate) using a single computer. Test output, compilation 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.

3.7.3 Test Site

Testing was conducted at Siemens AG, Munich and was completed on 6th March 1989.
Siemens AG has submitted the following Declaration of Conformance concerning the Siemens ES2000 Ada Compiler V2.0.
DECLARATION OF CONFORMANCE

DECLARATION OF CONFORMANCE

Compiler Implementor: SIEMENS AG, München, FRG
Ada Validation Facility: IABG mbH, 8012 Ottobrunn, FRG
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name: Siemens BS 2000 Ada Compiler V2.0
Host and Target Architecture: Siemens 7.590G
Host and Target OS and Version: Siemens BS2000/V9.0

Derived Compiler Registration

Derived Compiler Name: Same as above
Host and Target Architecture: 7.531, 7.536, 7.541, 7.551,
7.530, 7.550, 7.560,
7.561, 7.571, 7.550,
7.560, 7.570, 7.580, 7.590
7.700

Implementor’s Declaration

I, the undersigned representing SIEMENS AG, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that Siemens AG is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner’s corporate name.

Dr. W. Hoyer, Deputy Manager
Date: March 8, 1989

SIEMENS AG

I, the undersigned, representing SIEMENS AG, take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Dr. W. Hoyer, Deputy Manager
Date: March 8, 1989
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 the Siemens BS2000 Ada Compiler V2.0, as described in this Appendix, are provided by Siemens AG. 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 -2_147_483_649 .. 2_147_483_647;
type SHORT_INTEGER is range -32_678 .. 32_767;

type FLOAT is digits 15 range -2#1.0E212 .. 2#1.0E212;
type DURATION is delta 2#1.0E-14 range -131_071.0 .. 131_071.0;

-- DURATION'SMALL = 2#1.0E-14;

... end STANDARD;
F. Implementation-Dependent Characteristics

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

This appendix summarizes all implementation-dependent characteristics of the Siemens BS2000 Ada Compiler. It describes:

1. The form, allowed places, and effect of every implementation-dependent pragma.
2. The name and the type of every implementation-dependent attribute.
3. The specification of the package SYSTEM (see 13.7).
4. The list of all restrictions on representation clauses (see 13.1).
5. The conventions used for any implementation-generated name denoting implementation-dependent components (see 13.4).
6. The interpretation of expressions that appear in address clauses, including those for interrupts (see 13.5).
7. Any restriction on unchecked conversions (see 13.10.2).
8. Any implementation-dependent characteristics of the input-output packages (see 14).

F.1 Implementation-Dependent Pragmas

There are no implementation-defined pragmas.

The only language names accepted by pragma INTERFACE are ASSEMBLER and COBOL.

The only priority accepted by pragma PRIORITY is represented by an expression of the static value 0 (cf. the definition of the subtype PRIORITY in package SYSTEM).

F.2 Implementation-Dependent Attributes

There are no implementation-dependent attributes apart from those described in F.5.
Implementation-Dependent Characteristics

F.3 Specification of the Package SYSTEM

package SYSTEM is

    type ADDRESS is new INTEGER;
    type NAME is (BS2000);
    SYSTEM_NAME : constant NAME := BS2000;
    STORAGE_UNIT : constant := 8;
    MEMORY_SIZE : constant := 4000000;

    -- System-Dependent Named Numbers:
    MIN_INT : constant := -2147483648;
    MAX_INT : constant := 2147483647;
    MAX_DIGITS : constant := 15;
    MAX_MANTISSA : constant := 31;
    FINE_DELTA : constant := 2.0**(-30);
    TICK : constant := 0.0001;

    -- Other System-Dependent Declarations
    subtype PRIORITY is INTEGER range 0 .. 0;

end SYSTEM;

F.4 Restrictions on Representation Clauses

1. Length Clauses:
   For the specification of SIZE all values from 0 to INTEGER'LAST can be given. The length clause is
   accepted if the value specified is not smaller than the number of bits used to represent all possible
   values.

2. The following list shows the smallest values allowed for some types.
   boolean types  :  1 bit
   access types  :  32 bits
   task types    :  32 bits
   INTEGER       :  32 bits
   FLOAT         :  64 bits
   CHARACTER     :  8 bits
   DURATION       :  32 bits

Generally a representation clause for a derived type of a private type is rejected if the full declara-
tion is a composite type and if the derived type has to be packed to fulfill the representation clause.

4. Specification of SMALL for a fixed-point type: the specified value must be a power of 2; other
   values are not accepted.

5. Enumeration Representation Clauses:
   An enumeration representation clause is not accepted for a type derived from the predefined
   enumeration type BOOLEAN.

6. An enumeration representation clause is not accepted for a derived type if a constraint is given for
   the parent subtype associated with this derived type.
Implementation-Dependent Characteristics

7 An enumeration representation clause is not accepted if any of the internal codes does not lie within the range SYSTEM.MIN_INT to SYSTEM.MAX_INT.

8 Component Clauses:
In addition to the restrictions stated in the LRM 13.4 the following implementation-dependent restrictions on the values of the static expressions given in a component clause exist:

9 Consider the component clause

\[
\text{COMPONENT at EXP range LOW..UPP}
\]

10 Define
\[
\begin{align*}
\text{component offset} & : = \text{EXP} \times \text{SYSTEM.STORAGE\_UNIT} + \text{LOW}; \\
\text{component size} & : = \max ( \text{UPP} - \text{LOW} + 1, 0 );
\end{align*}
\]

11 Both values for component offset and component size must lie in the range 0..SYSTEM.MAX_INT; otherwise, the component clause is not accepted. Furthermore, the following restrictions depending on the type associated with the corresponding component apply:

12 (1) for components of some discrete type the following restrictions apply:
\[
\begin{align*}
\text{component offset} & : \text{no further restrictions.} \\
\text{component size} & : \text{must be less than or equal to 32.}
\end{align*}
\]

13 (2) for components of some fixed point type the following restrictions apply:
\[
\begin{align*}
\text{component offset} & : \text{no further restrictions.} \\
\text{component size} & : \text{must be less than or equal to 32.}
\end{align*}
\]

14 (3) for components of some floating point type the following restrictions apply:
\[
\begin{align*}
\text{component offset} & : \text{must be a multiple of 64.} \\
\text{component size} & : \text{must equal 64.}
\end{align*}
\]

15 (4) for components of some access type the following restrictions apply:
\[
\begin{align*}
\text{component offset} & : \text{must be a multiple of 32.} \\
\text{component size} & : \text{must equal 32.}
\end{align*}
\]

16 (5) for components of some task type the following restrictions apply:
\[
\begin{align*}
\text{components offset} & : \text{must be a multiple of 32.} \\
\text{components size} & : \text{must equal 32.}
\end{align*}
\]

17 (6) for components of some array type the following restrictions apply:
\[
\begin{align*}
\text{component offset} & : \text{must be a multiple of the alignment of the array elements in terms of bits.} \\
\text{component size} & : \text{must equal the size of the array type.}
\end{align*}
\]

18 (7) for components of some record type the following restrictions apply:
\[
\begin{align*}
\text{component offset} & : \text{must be a multiple of the maximum over the alignments of the record components and a multiple of the user-specified alignment (if any) for the record type in terms of bits.} \\
\text{component size} & : \text{must equal the (maximum) size of the record type.}
\end{align*}
\]

19 Alignment clauses:
An alignment given in an alignment clause is accepted only if the value (in terms of storage units) does lie within the range 1..8.
An alignment given in an alignment clause is accepted only if the value (in terms of bits) is a multiple of the maximum over the alignments of all components of the corresponding record type.

F.5 Conventions for Implementation-Generated Names Denoting Implementation-Dependent Components in Record Representation Clauses

Implementation-dependent components may be added to record objects by the compiler.

Storage place specification for implementation-dependent record components can be achieved solely by means of implementation-dependent attributes. The following list shows all implementation defined attributes and the restrictions applied to the corresponding implementation dependent components.

For a record type R:

- **R' RECORD _ SIZE**: For any record object of type R this implementation-defined attribute denotes an implicit component which yields the number of bits allocated for the record object. The implementation-dependent component exists only if the objects of type R may vary in size.

- **R' VARIANT _ INDEX**: For any record object of type R this implementation-defined attribute denotes an implicit component which yields a value used to determine which components exist for the given record object. The implementation-dependent component exists only if R is a record type with a variant part.

For a component C of a record type R:

- **C' DOPE**: For any record object of type R this implementation-defined attribute denotes an implicit component which yields the offset of the component C relative to the start of the record. The implementation-dependent component exists only if the component C is of a component type whose constraint is dynamic.

- **C' ARRAY _ DESCRIPTOR**: For any record object of type R this implementation-defined attribute denotes an implicit component which yields values used to describe the component C. The implementation-dependent component exists only if C is of an array type and if the component subtype definition for C includes an index constraint which contains at least one discriminant of R.

- **C' RECORD _ DESCRIPTOR**: For any record object of type R this implementation-defined attribute denotes an implicit component which yields values used to describe the component C.
Implementation-Dependent Characteristics

The implementation-dependent component exists only if C is of a record type and if the component subtype definition for C includes a discriminant constraint which contains at least one discriminant of R.

Any of these implementation-defined attributes can be used only as component names in component clauses. Any other use of these attributes is illegal.

Any of these implementation-defined attributes can be used only if the corresponding implementation-dependent component exists for the corresponding record type or record component, respectively; otherwise, the corresponding component clause is not accepted.

The following restrictions apply to the component offset and component size for the above-defined implementation-dependent components in corresponding component clauses:

**RECORD SIZE:**
- component offset: must be a multiple of 32.
- component size: must equal 32.

**VARIANT_INDEX:**
- component offset: must be a multiple of 16.
- component size: must equal 16.

**DOPE:**
- component offset: must be a multiple of 16.
- component size: must equal 16.

**ARRAY_DESCRIPTOR:**
- component offset: must be a multiple of 32.
- component size: must equal 32 + n, if n is the number of dimensions of the array type associated with the corresponding component.

**RECORD_DESCRIPTOR:**
- component offset: must be a multiple of 32.
- component size: must equal 32.

F.6 Interpretation of Expressions Appearing in Address Clauses

Address clauses are not yet supported.

F.7 Restrictions on Unchecked Type Conversions

The Siemens BS2000 Ada compiler supports the generic function UNCHECKED_CONVERSION with the following restriction:
Implementation-Dependent Characteristics

The actual generic subtype corresponding to the formal generic type TARGET must not be an unconstrained array type, and it must not be an unconstrained type with discriminants that have no defaults. The instances gained from UNCHECKED_CONVERSION return a target value whose bit pattern is a left-aligned copy of that of the source value. The number of bits transferred corresponds to the size of the target subtype. If the size of the source value is greater than the size of the target subtype, then the source value information is truncated on the right hand side, i.e. the low order bits are ignored. If the size of the source value is not greater than the size of the target subtype, then - again - as many bits are transferred as corresponds to the size of the target subtype, and no padding with zeroes, spaces or other characters is performed.

F.8 Implementation-Dependent Characteristics of the Input-Output Packages

F.8.1 Introduction

The SEQUENTIAL_IO, DIRECT_IO and TEXT_IO packages are all written in Ada and they make calls on BASIC_IO which is a "typeless" package working with addresses and byte counts. SEQUENTIAL_IO and DIRECT_IO are generic packages, also INTEGER_IO, FIXED_IO, FLOAT_IO and ENUMERATION_IO in TEXT_IO are generic.

The routines written in assembler language have the name ADARTSBx with x in 1 .. 9, A .. Q, while the BASIC_IO routines have nearly the same names as in the input-output packages of Ada.

F.8.2 Conventions for NAME and FORM

External files are supported by the SAM, ISAM, SYSDTA, SYSOUT and SYSLST BS2000 files where the value of the parameter FORM of the CREATE and OPEN procedures determines which access method is selected.

The set of allowable values of FORM is given below together with the type of BS2000 file corresponding to it. Leading blanks and lower-case letters are not allowed in the FORM string.

<table>
<thead>
<tr>
<th>value of FORM BS2000 access method</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
</tr>
<tr>
<td>ISAM</td>
</tr>
<tr>
<td>SYSDTA</td>
</tr>
<tr>
<td>SYSOUT</td>
</tr>
<tr>
<td>SYSLST</td>
</tr>
<tr>
<td>SAM_PRINT</td>
</tr>
<tr>
<td>ISAM_PRINT</td>
</tr>
<tr>
<td>PAM</td>
</tr>
<tr>
<td>EAM</td>
</tr>
<tr>
<td>OMF</td>
</tr>
</tbody>
</table>

Each input-output package operates on a subset of the allowable forms.
SAM, ISAM, SAM_PRINT, ISAM PRINT and PAM files are identified by the value of the parameter NAME of the CREATE and OPEN procedures whose characters must conform to the BS2000 file naming conventions as described below. The value of the parameter NAME is ignored for other values of FORM.

The syntax associated with the string NAME is as follows:

```plaintext
NAME ::= .link_name | file_name
file_name ::= :cat_id: $user_id . name {. name } |
             $user_id . name {. name } |
             $admin_name |
             name {. name }

cat_id ::= name_character
link_name ::= name_character { name_character }
user_id ::= name_character { name_character }
admin_name ::= name_character { name_character }
name ::= name_character { name_character }

name_character ::= upper_case_letter | digit |
                 special_character

special_character ::= $ | @ | # | -
```

BS2000 imposes the following additional restrictions upon the syntax of NAME:

1. The maximum length of a link_name or a user_id is eight characters.
2. The maximum length of a file_name starting with :cat_id: is 54 characters.
3. The maximum length of a file_name starting with $user_id is 51 characters.
4. The maximum length of an admin_name is 47 characters unless it contains one or more periods in which case the maximum length is 53 characters.
5. The maximum length of a file_name starting with name is 41 characters.
6. The first character of a name must not be a special character, and the last character must not be a hyphen.
7. A file_name must include at least one letter.
Example of using TEXT.IO:

with TEXT.IO; use TEXT.IO;
package FILE_MANAGEMENT is

   ACTUAL_FILE1 : TEXT.IO.FILETYPE;
   ACTUAL_FILE2 : TEXT.IO.FILETYPE;
   ACTUAL_FILE3 : TEXT.IO.FILETYPE;

begin
   -- Create a BS2000-SAM file with name A.SAM.FILE
   TEXT.IO.CREATE (FILE => ACTUAL_FILE1,
                    MODE => OUT_FILE,
                    NAME => "A.SAM.FILE",
                    FORM => "SAM");

   -- Create a BS2000-ISAM file with the link name ABCD and with file_name
   -- AN.ISAM.FILE
   -- BS2000 command: "/FILE AN.ISAM.FILE.LINK = ABCD" (Note: no ")."
   TEXT.IO.CREATE (FILE => ACTUAL_FILE2,
                    MODE => OUT_FILE,
                    NAME => ".ABCD",  -- Note: with "."
                    FORM => "ISAM");

   -- Open the BS2000-SAM file with link_name XYZ and with file_name A.SAM.FILE
   -- BS2000 command: "/FILE A.SAM.FILE.LINK = XYZ" (Note: no ")."
   TEXT.IO.OPEN (FILE => ACTUAL_FILE3,
                  MODE => IN_FILE,
                  NAME => ".XYZ",  -- Note: with "."
                  FORM => "SAM");
end FILE_MANAGEMENT;

F.8.3 File management

This section describes the implementation restrictions which apply to the sequential, direct and text input-output packages equally. The maximum number of objects which may be stored in an external file is dependent upon the maximum number of records or the maximum number of blocks which may be stored in its underlying BS2000 file. The values given below state this maximum for each FORM provided that limits imposed by the system configuration are not otherwise reached. For the direct and sequential input-output packages, each object is stored in a separate record or block; for the text input-output package, each line is stored in a separate record.

<table>
<thead>
<tr>
<th>FORM</th>
<th>Maximum Number of Records / Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
<td>configuration dependent limit</td>
</tr>
<tr>
<td>ISAM</td>
<td>99 999 999 records</td>
</tr>
<tr>
<td>SAM PRINT</td>
<td>configuration dependent limit</td>
</tr>
</tbody>
</table>
Implementation-Dependent Characteristics

<table>
<thead>
<tr>
<th>File Format</th>
<th>Records</th>
<th>Configuration Dependent Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISAM_PRINT</td>
<td>99 999 999</td>
<td>configuration dependent limit</td>
</tr>
<tr>
<td>PAM</td>
<td></td>
<td>configuration dependent limit</td>
</tr>
<tr>
<td>EAM</td>
<td></td>
<td>configuration dependent limit</td>
</tr>
<tr>
<td>OMF</td>
<td></td>
<td>configuration dependent limit</td>
</tr>
<tr>
<td>SYSDTA</td>
<td></td>
<td>configuration dependent limit</td>
</tr>
<tr>
<td>SYSOUT</td>
<td></td>
<td>configuration dependent limit</td>
</tr>
<tr>
<td>SYSLST</td>
<td></td>
<td>configuration dependent limit</td>
</tr>
</tbody>
</table>

Two alternative record formats are available for ISAM and SAM files, varying and constant length. TEXT.IO always uses varying length records whereas DIRECT.IO and SEQUENTIAL.IO support both formats. A varying length record format is used if an instance of direct or sequential input-output packages uses unconstrained element-types. Otherwise a fixed length record format is used where the length equals the value of \((\text{ELEMENT}\ \text{TYPE}\cdot\text{SIZE} + 7) / 8\) (that are the number of bytes needed for this special type).

The maximum size of the objects which can be stored in an external file is restricted. The universal integer value which results from the application of the \text{SIZE} attribute to every object accessed by the package must lie within a range which is dependent upon the \text{FORM} and whether constant or varying size records are being used. The exception USE_ERROR is raised if this constraint is violated.

<table>
<thead>
<tr>
<th>Form</th>
<th>Constant/Varying</th>
<th>Object Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM</td>
<td>constant</td>
<td>1..16384</td>
</tr>
<tr>
<td>SAM</td>
<td>varying</td>
<td>1..16352</td>
</tr>
<tr>
<td>SAM_PRINT</td>
<td>varying</td>
<td>1..16352</td>
</tr>
<tr>
<td>ISAM</td>
<td>constant</td>
<td>1..16320</td>
</tr>
<tr>
<td>ISAM</td>
<td>varying</td>
<td>1..16288</td>
</tr>
<tr>
<td>ISAM_PRINT</td>
<td>varying</td>
<td>1..16288</td>
</tr>
<tr>
<td>PAM</td>
<td>constant/varying</td>
<td>1..16368</td>
</tr>
<tr>
<td>EAM</td>
<td>constant/varying</td>
<td>1..16368</td>
</tr>
<tr>
<td>OMF</td>
<td>constant</td>
<td>16368</td>
</tr>
<tr>
<td>SYSDTA</td>
<td>varying</td>
<td>1..2032</td>
</tr>
<tr>
<td>SYSOUT</td>
<td>varying</td>
<td>1..2032</td>
</tr>
<tr>
<td>SYSLST</td>
<td>varying</td>
<td>1..2032</td>
</tr>
</tbody>
</table>

The default value in TEXT.IO for the \text{FORM} parameter is SAM_PRINT, in SEQUENTIAL.IO it is SAM, while in DIRECT.IO it is ISAM.

SAM and ISAM files with no null string for NAME are permanent files in that their lifetimes are independent of the currently running Ada program and of the BS2000 tasks in which they were created. Permanent files may be closed in one BS2000 task and opened subsequently in the same or another task without loss of their contents (for \text{MODE} = \text{IN} or \text{IN-OUT}).

A null string for NAME specifies an external file that is not accessible after the completion of the main program (a temporary file).

The BS2000 names for temporary files are

"#ADA.xxxx.yymmdd.zzzzzz.nnnnn" with

- xxxx ::= decimal number (tsn of the current BS2000 task)
- yymmdd ::= decimal number (current date)
- zzzzzz ::= decimal number (time in seconds filled up with leading 0)
- nnnnn ::= decimal number (range 10000..99999).
Implementation-Dependent Characteristics

10 When a SAM or ISAM file (selected by the value of NAME) is opened, there is no check that the form of the BS2000 file corresponds to the value of the FORM parameter of the OPEN procedure. There is no check either that the input-output package opening a SAM or ISAM file is the same package as the one which created the file. If either of these conditions is violated, the program may deliver unexpected results.

11 PAM is the primary block-oriented access method of BS2000 to enable random access to blocks of 2048 bytes.

12 EAM is the access method to temporary files of BS2000. It enables random access to blocks of 2048 bytes. In Sequential.IO and Direct.IO these files are only accessible within one Ada-Program. A string for NAME is ignored.

13 OMF is the access method to generate object modules. An OMF-files is a special form of an EAM-file.

14 SYSSTA, SYSOUT and SYSLST files are temporary files whose lifetime ends with that of the BS2000 task which created them.

The SYSSTA, SYSOUT and SYSLST files are unique within a BS2000 task. SYSSTA and SYSOUT are opened by the elaboration of TEXT.IO. SYSSTA is the FORM of the Ada STANDARD_INPUT file, while SYSOUT is the FORM of the Ada STANDARD_OUTPUT file. The user may open SYSSTA at most once at a time additionally to STANDARD_INPUT. Also only one file may be opened at a time with FORM parameter SYSLST. Opening a file with these FORM parameters causes a SYSFILE command for the BS2000 system. Therefore, reading from the BS2000 system file SYSSTA is equivalent to reading from STANDARD_INPUT, but both have their own FCB. A close on a file with the FORM parameter SYSSTA or SYSLST causes a redirection of the BS2000 system files SYSSTA and SYSLST to (PRIMARY) via a SYSFILE command. The user may redirect SYSSTA or SYSLST to (PRIMARY), too, by opening a file with the FORM parameter SYSSTA or SYSLST and NAME parameter (PRIMARY). A SYSOUT file may not be opened or deleted because its redirection is impossible and STANDARD_OUTPUT is opened with the FORM parameter SYSOUT during the elaboration.

16 No assumptions should be made about the way objects are stored in the various BS2000 files except as described for the TEXT.IO package. For example, the mapping of indices onto ISAM keys may differ between different versions of the input-output packages.

F.8.3.1 SEQUENTIAL_IO

1 The value of the FORM parameter of the CREATE and OPEN procedures is restricted to SAM, PAM, EAM and OMF.

2 The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and unconstrained record types with discriminants that have no defaults.

F.8.3.2 DIRECT_IO

1 The value of the FORM parameter of the CREATE and OPEN procedures is restricted to ISAM, PAM, EAM and OMF.
Implementation-Dependent Characteristics

The package DIRECT_IO cannot be instantiated with unconstrained array types and unconstrained record types with discriminants that have no defaults.

The value of an index may be set in the range 1 .. INTEGER'LAST.

F.8.3.3 TEXT_IO

The value of the FORM parameter of the CREATE procedure is restricted to SAM, ISAM, SAM_PRINT and ISAM_PRINT, while the OPEN procedure may use SYSLST and SYSDTA additionally.

The lines contained in text files are variable in length in the range 1 .. 1980 characters. The upper bound for the subtype FIELD is 500;

The upper bound for the type COUNT is 1980.

In printable files (FORM = SAM_PRINT, FORM = ISAM_PRINT) lines are stored in the second to 2001st character of a BS2000 variable length file record. The ASCII characters of the Ada program are represented by their corresponding EBCDIC characters in the BS2000 files. The first character of the record is a printer control character where ' ' means line-feed and 'A' page feed. Thus BS2000 files created by a call to TEXT_IO can be printed using the /PRINT command (with SPACE = E) or displayed using the EDOR and EDT text file editors. The printer control characters are used to implement the line and page terminators and can be manipulated using the standard line and page control procedures. The transfer of lines to BASIC_IO is done by NEW_LINE, NEW_PAGE, CLOSE and RESET - or if a line is filled up.

An empty line after a page terminator is identified by an EBCDIC.NUL in the second column. Other empty lines are identified by an EBCDIC.SOH in the first and a EBCDIC.BLANK in the second column. Since TEXT_IO converts ASCII.NUL to EBCDIC.NUL this special character may not be used in the first column of the first line of a new page as the only character in this line (that is the second column of the BS2000 file).

Two FORM parameters (SAM and ISAM) may be used by TEXT_IO to support files without printer control characters in the first column.

In these files the end of a line is interpreted as a line terminator. A page terminator is an EBCDIC.NUL in the last column of the line. Therefore the user may not output an EBCDIC.NUL to the last column of a line without an incrementation of the current page by TEXT_IO on reading the file again. An EBCDIC.STX in the first column designates an empty line.

The MODE of the SYSLST file is restricted to OUT_FILE.
The MODE of the SYSDTA file is restricted to IN_FILE.

The standard input file has the FORM SYSDTA and the standard output file has the FORM SYSOUT.

In the dialogue mode of BS2000 a call of NEW_PAGE on STANDARD_OUTPUT causes the deletion of the screen and a call of NEW_LINE with an empty internal buffer causes the output of a line feed.

The transfer of characters from TEXT_IO to BASIC_IO is done line by line. All characters are stored in an internal buffer. The line is displayed after calling PUT_LINE, NEW_LINE, NEW_PAGE, CLOSE and RESET. On the other hand the terminal represents the two distinct files STANDARD_OUTPUT and STANDARD_INPUT in one "file" (terminal). Therefore, a sequence of PUT - GET - PUT routine calls without calling NEW_LINE or NEW_PAGE causes the following display sequence at the terminal.
Implementation-Dependent Characteristics

Example:

```pascal
with TEXT_IO; use TEXT_IO;
package DIALOGUE is
begin

    PUT (STANDARD_OUTPUT, " THE USER AT THE TERMINAL IS ");
    GET (STANDARD_INPUT, USER_NAME); -- the user enters "TOM WHO IS NOT"
    PUT (STANDARD_OUTPUT, " CRAZY");
    NEW LINE (STANDARD_OUTPUT);

end DIALOGUE;
```

Example of the interaction (characters typed by the user are italicized):

-- the user has to enter something, assume he enters "TOM WHO IS NOT"

* TOM WHO IS NOT  
THE USER AT THE TERMINAL IS CRAZY

The user intended to get:

THE USER AT THE TERMINAL IS
* TOM WHO IS NOT  
CRAZY

The user should use in those cases a sequence of PUTLINE - GET - PUT - NEW LINE. Then the example
would display:

THE USER AT THE TERMINAL IS
* TOM WHO IS NOT  
CRAZY

A text file read from a terminal (via SYS$TA) is handled like a file with FORM SAM, that means page
terminators and empty lines are recognized by an EBCDIC NUL or EBCDIC STX as described above for
SAM files.
Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below. The use of the operator '*\' signifies a multiplication of the following character. The use of the '&\' character signifies concatenation of the preceding and following strings. The values within single or double quotation marks are to highlight characters or string values:

<table>
<thead>
<tr>
<th>Name and Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ACC_SIZE</td>
<td>32</td>
</tr>
<tr>
<td>An integer literal whose value is the number of bits sufficient to hold any value of an access type.</td>
<td></td>
</tr>
<tr>
<td>$BIG_ID1</td>
<td>239 'A' &amp; '1'</td>
</tr>
<tr>
<td>An identifier the size of the maximum input line length which is identical to $BIG_ID2 except for the last character.</td>
<td></td>
</tr>
<tr>
<td>$BIG_ID2</td>
<td>239 'A' &amp; '2'</td>
</tr>
<tr>
<td>An identifier the size of the maximum input line length which is identical to $BIG_ID1 except for the last character.</td>
<td></td>
</tr>
<tr>
<td>$BIG_ID3</td>
<td>120 'A' &amp; '3' &amp; 119 'A'</td>
</tr>
<tr>
<td>An identifier the size of the maximum input line length which is identical to $BIG_ID4 except for a character near the middle.</td>
<td></td>
</tr>
</tbody>
</table>
TEST PARAMETERS

Name and Meaning

BIG_ID4
An identifier the size of the maximum input line length which
is identical to BIG_ID3 except for a character near the middle.

BIG_INT_LIT
An integer literal of value 298
with enough leading zeroes so
that it is the size of the
maximum line length.

BIG_REAL_LIT
A universal real literal of
value 690.0 with enough leading
zeroes to be the size of the
maximum line length.

BIG_STRING1
A string literal which when
catenated with BIG_STRING2
yields the image of BIG_ID1.

BIG_STRING2
A string literal which when
catenated to the end of
BIG_STRING1 yields the image of
BIG_ID1.

BLANKS
A sequence of blanks twenty
characters less than the size
of the maximum line length.

COUNT_LAST
A universal integer
literal whose value is
TEXT_10.COUNT'LAST.

DEFAULT_MEM_SIZE
An integer literal whose value
is SYSTEM.MEMORY_SIZE.

DEFAULT_STOR_UNIT
An integer literal whose value
is SYSTEM.STORAGE_UNIT.

Value

120 * 'A' & '4' & 119 * 'A'

237 * '0' & "298"

235 * '0' & "690.C"

"" & 120 * 'A' & ""

"" & 119 * 'A' & '1' & ""

220 * ""

1980

4_000_000

8
### Name and Meaning

<table>
<thead>
<tr>
<th>Name and Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DEFAULT_SYS_NAME</td>
<td>The value of the constant SYSTEM.SYSTEM_NAME.</td>
</tr>
<tr>
<td>$DELTA_DOC</td>
<td>A real literal whose value is SYSTEM.FINE_DELTA.</td>
</tr>
<tr>
<td>$FIELD_LAST</td>
<td>A universal integer literal whose value is TEXT_IO.FIELD'LAST.</td>
</tr>
<tr>
<td>$FIXED_NAME</td>
<td>The name of a predefined fixed-point type other than DURATION.</td>
</tr>
<tr>
<td>$FLOAT_NAME</td>
<td>The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.</td>
</tr>
<tr>
<td>$GREATER_THAN_DURATION</td>
<td>A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.</td>
</tr>
<tr>
<td>$GREATER_THAN_DURATION_BASE_LAST</td>
<td>A universal real literal that is greater than DURATION'BASE'LAST.</td>
</tr>
<tr>
<td>$HIGH_PRIORITY</td>
<td>An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.</td>
</tr>
<tr>
<td>$ILLEGAL_EXTERNAL_FILE_NAME1</td>
<td>An external file name which contains invalid characters.</td>
</tr>
<tr>
<td>$ILLEGAL_EXTERNAL_FILE_NAME2</td>
<td>An external file name which is too long.</td>
</tr>
<tr>
<td>$INTEGER_FIRST</td>
<td>A universal integer literal whose value is INTEGER'FIRST.</td>
</tr>
</tbody>
</table>
# TEST PARAMETERS

<table>
<thead>
<tr>
<th>Name and Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER'LAST</td>
<td>2147483647</td>
</tr>
<tr>
<td>A universal integer literal whose value is INTEGER'LAST.</td>
<td></td>
</tr>
<tr>
<td>INTEGER'LAST+1</td>
<td>2147483648</td>
</tr>
<tr>
<td>A universal integer literal whose value is INTEGER'LAST + 1.</td>
<td></td>
</tr>
<tr>
<td>LESS_THAN_DURATION</td>
<td>-131071.5</td>
</tr>
<tr>
<td>A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.</td>
<td></td>
</tr>
<tr>
<td>LESS_THAN_DURATIION_BASE_FIRST</td>
<td>-200000.0</td>
</tr>
<tr>
<td>A universal real literal that is less than DURATION'BASE'FIRST.</td>
<td></td>
</tr>
<tr>
<td>LOW_PRIORITY</td>
<td>0</td>
</tr>
<tr>
<td>An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.</td>
<td></td>
</tr>
<tr>
<td>MANTISSA_DOC</td>
<td>31</td>
</tr>
<tr>
<td>An integer literal whose value is SYSTEM.MAX_MANTISSA.</td>
<td></td>
</tr>
<tr>
<td>MAX_DIGITS</td>
<td>15</td>
</tr>
<tr>
<td>Maximum digits supported for floating-point types.</td>
<td></td>
</tr>
<tr>
<td>MAX_IN_LEN</td>
<td>240</td>
</tr>
<tr>
<td>Maximum input line length permitted by the implementation.</td>
<td></td>
</tr>
<tr>
<td>MAX_INT</td>
<td>2147483647</td>
</tr>
<tr>
<td>A universal integer literal whose value is SYSTEM.MAX_INT.</td>
<td></td>
</tr>
<tr>
<td>MAX_INT+1</td>
<td>2147483648</td>
</tr>
<tr>
<td>A universal integer literal whose value is SYSTEM.MAX_INT+1.</td>
<td></td>
</tr>
<tr>
<td>MAX LEN_INT_BASED_LITERAL</td>
<td>&quot;2:&quot; &amp; 235 * '0' &amp; &quot;1:&quot;</td>
</tr>
<tr>
<td>A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.</td>
<td></td>
</tr>
</tbody>
</table>
TEST PARAMETERS

Name and Meaning

$MAX_LEN_REAL_BASED_LITERAL
A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.

$MAX_STRING_LITERAL
A string literal of size MAX_IN_LEN, including the quote characters.

$MIN_INT
A universal integer literal whose value is SYSTEM.MIN_INT.

$MIN_TASK_SIZE
An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.

$NAME
A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.

$NAME_LIST
A list of enumeration literals in the type SYSTEM.NAME, separated by commas.

$NEG_BASED_INT
A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.

$NEW_MEM_SIZE
An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.

Value

"16:" & 233 * '0' & "F.E:"

"" & 118 * 'A' & ""

-2147483648

32

LONG_LONG_INTEGER

B52000

8#37777777776#

4_000_000

C-5
### Name and Meaning

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NEW_STOR_UNIT</em></td>
<td>8</td>
</tr>
<tr>
<td>An integer literal whose value is a permitted argument for pragma STORAGE_UNIT, other than <em>DEFAULT_STOR_UNIT</em>. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.</td>
<td></td>
</tr>
<tr>
<td><em>NEW_SYS_NAME</em></td>
<td>BS2000</td>
</tr>
<tr>
<td>A value of the type SYSTEM.NAME, other than <em>DEFAULT_SYS_NAME</em>. If there is only one value of that type, then use that value.</td>
<td></td>
</tr>
<tr>
<td><em>TASK_SIZE</em></td>
<td>32</td>
</tr>
<tr>
<td>An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.</td>
<td></td>
</tr>
<tr>
<td><em>TICK</em></td>
<td>0.0001</td>
</tr>
<tr>
<td>A real literal whose value is SYSTEM.TICK.</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX D
WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 43 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-dddd is to an Ada Commentary.

a. E28005C This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

b. A39005G This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

c. B97102E This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

d. BC3009B This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

e. CD2A62D This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

f. CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests] These tests wrongly attempt to check the sizes of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.
WITHDRAWN TESTS

g. CD2A81G, CD2A83G, CD2A84N & M, & CD50110 [5 tests] These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

h. CD2B15C & CD7205C These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

i. CD2D11B This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

j. CD5007B This test wrongly expects an implicitly declared subprogram to be at the the address that is specified for an unrelated subprogram (line 303).

k. ED7004B, ED7005C & D, ED7006C & D [5 tests] These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

l. CD7105A This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK—particular instances of change may be less (line 29).

m. CD7203B, & CD7204B These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

n. CD7205D This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

o. CE2107I This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

p. CE3111C This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.
q. CE3301A This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

r. CE3411B This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.
Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 89030611.10059
Siemens AG
Siemens BS2000 Ada Compiler V2.0
Siemens 7.590G Host and Target

Completion of On-Site Testing:
6th March 1989

Prepared By:
IABG mbH, Abt SZT
Einsteinstr 20
D-8012 Ottobrunn
West Germany

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081
COMOPT  Festlegen der Compileroptionen


Beim Einrichten einer Bibliothek gelten folgende Voreinstellungen:

```
ADDSOURCE  = YES
DEBUG      = NO
KEEPXREF   = NO
LISTMODE   = (NOOBJECT, NOOPTIONS,
              SOURCE, NOSTORAGE, NOXREF)
OPTIMIZE   = NO
RUNMODE    = BATCH
```

```
comopt_cmd ::= 
  COMOPT [(comopt_param{, comopt_param})];
comopt_param ::= 
  ADDSOURCE = yes_or_no
  DEBUG     = debugmode
  KEEPXREF  = yes_or_no
  LISTMODE  = listmode(listmode)
  OPTIMIZE  = yes_or_no
  RUNMODE   = runmode

dbgmode ::= 
  NO | R[ESTRICTED] | F[ULL]

listmode ::= 
  OBJ[ECT] | NOOBJ[ECT]
  OPT[IIONS] | NOOPT[IIONS]
  S[OURCE]  | NOST[OURCE]
  S[TORAGE] | NOST[ORAGE]
  X[REF]    | NOX[REF]

runmode ::=  B[ATCH] | D[IALOGUE]
```

-------------------------------------- -----------------------------------------
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDSOURCE</td>
<td>Dieser Parameter aktiviert oder deaktiviert die Quelltextverwaltung bei fehlerfreien Übersetzungen.</td>
</tr>
<tr>
<td>= YES</td>
<td>Der (fehlerfreie) Quelltext und das Compilerlisting werden unter der Ada-Einheit verwaltet. Das Quellfragment bleibt unverändert in der Bibliothek. Die Quelltext-Verwaltung ist nur aktiv, wenn ein Quellfragment genau eine Ada-Übersetzungseinheit enthält und der Name des Fragments mit dem Namen dieser Übersetzungseinheit übereinstimmt; andernfalls wird die Quelltextverwaltung automatisch deaktiviert.</td>
</tr>
<tr>
<td>= NO</td>
<td>Die Quelltextverwaltung ist nicht aktiv; das Compilerlisting und die Quelle einer Ada-Einheit sind nur als Fragmente verfügbar.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Gibt an, mit welchen Debugger-Anweisungen die übersetzte Einheit getestet werden kann.</td>
</tr>
<tr>
<td>= NO</td>
<td>Die Übersetzungseinheit kann nicht mit dem Debugger getestet werden.</td>
</tr>
<tr>
<td>= RESTRICTED</td>
<td>Es ist nur &quot;passives&quot; Testen durch den Debugger möglich, d.h. es können wohl Haltepunkte gesetzt und Variablenwerte ausgegeben werden, es ist aber nicht zugelassen, den Inhalt der Variablen zu verändern.</td>
</tr>
<tr>
<td>= FULL</td>
<td>Alle Debugger-Anweisungen werden unterstützt. Der Modus ist nur mit OPTIMIZE = NO kompatibel.</td>
</tr>
<tr>
<td>KEEPXREF</td>
<td>Es wird Information in der Bibliothek abgelegt, die das Erstellen einer globalen Querverweisliste ermöglicht.</td>
</tr>
<tr>
<td>= YES</td>
<td></td>
</tr>
<tr>
<td>= NO</td>
<td>Es wird keine Information für ein Querverweislisting abgelegt.</td>
</tr>
<tr>
<td>LISTMODE</td>
<td>Definiert den Umfang des erzeugten Compilerlistings. Es kann gesteuert werden, ob daneben enthalten sein sollen:</td>
</tr>
<tr>
<td></td>
<td>- der Quelltext (SOURCE)</td>
</tr>
<tr>
<td></td>
<td>- der Objektcode (OBJECT)</td>
</tr>
<tr>
<td></td>
<td>- das Speicher Abbild der Variablen (STORAGE)</td>
</tr>
<tr>
<td></td>
<td>- die Querverweisliste (XREF)</td>
</tr>
<tr>
<td></td>
<td>- die Compileroptionen (OPTIONS)</td>
</tr>
<tr>
<td></td>
<td>Das erzeugte Listing enthält bei einer fehlerhaften Übersetzung immer den Quelltext mit den Fehlermeldungen.</td>
</tr>
<tr>
<td>OPTIMIZE</td>
<td>Es wird eine Optimierung zum Eliminieren von unnötigen Laufzeit-Checks eingeschaltet. Diese Optimierung ist nicht verträglich mit der Debuggeroption DEBUG = FULL.</td>
</tr>
<tr>
<td>= YES</td>
<td></td>
</tr>
<tr>
<td>= NO</td>
<td>Es wird keine Optimierung durchgeführt.</td>
</tr>
</tbody>
</table>
LINKOPT Festlegen der Binderoptionen

Mit der LINKOPT-Anweisung können für die Arbeitsbibliothek die Binderoptionen festgelegt werden. Beim Einrichten der Bibliothek werden gewisse Standardwerte eingestellt, die dann mit der LINKOPT-Anweisung modifiziert werden können. Alle Parameter dieser Anweisung können auch bei der LINK-Anweisung verwendet werden, sie gelten dann aber nur für die Dauer des Bindens. Die LINKOPT-Anweisung verändert nur die Voreinstellungen, die explizit genannt wurden. Werden keine Parameter angegeben, so werden die aktuellen Werte am Bildschirm angezeigt.

Das Binder-Listing wird standardmäßig in die Datei LL.main_program, die erzeugte Phase in die Datei GO.main_program geschrieben. Der Name main_program leitet sich aus dem Namen main_unitname des Hauptprogramms ab. Bei der Transformation des Namens wird der Unterstrich durch einen Bindestrich ersetzt. Namen mit mehr als 35 Zeichen werden abgeschnitten.

Der Parameter RUNMODE legt fest, ob das Binden im Dialog oder als Batch-Prozeß durchgeführt wird. Im Fall RUNMODE = DIALOGUE wird das Ada-Binden im Dialog durchgeführt, und die Steueranweisungen an den BS2000-Binder werden in die BS2000-Datei LL.main_program geschrieben. Der BS2000-Binder kann anschließend im Nebenprozeß oder nach Beenden der Ada-Programmierumgebung mit "DO BS2000-Binder" aufgerufen werden. Im voreingestellten Fall RUNMODE = BATCH werden das Ada-Binden und das BS2000-Binden zusammen im Batch-Prozeß durchgeführt.

Standardmäßig gilt:

```
DEBUG = NO
INSTRFILE = (DEFAULT)
LISTFILE = (DEFAULT)
LISTMODE = (NOELAB, NOMAP, NOOPT, NOSORT, NOUNITS, NOXREF)
PHASEFILE = (DEFAULT)
RESOLVELIB = (NONE)
RUNMODE = BATCH
KS = "O"
```

```
LINKOPT = linkopt_param (linkopt_param),

linkopt_param ::= =

D[EBUG] = debugmode 1
INSTRF[I]LE = instrfile 1
LISTF[I]LE = listfile 1
LISTM[ODE] = (link_listmode (.link_listmode)) 1
PHASEF[ILE] = phasefile 1
```
RESOLVE[LIB]  =  resolvelib
RUN[MODE]   =  runmode
XS           =  yes_or_no

debugmode   ::=  N(O)  |  R[ESTRICTED]  |  F[ULL]
instrfile   ::=  filename  |  (DEFAULT)
link_listmode ::=  E[LAB]  |  NOE[LAB]  |
               |  M[AP]  |  NOM[AP]  |
               |  OPT[IONS]  |  NOOPT[IONS]  |
               |  S[ORT]  |  NOS[ORT]  |
               |  U[NITS]  |  NOU[NITS]  |
               |  X[REF]  |  NOX[REF]  |
listfile    ::=  filename  |  (DEFAULT)
phasefile   ::=  filename  |  (DEFAULT)
resolvelib  ::=  filename  |  (NONE)
runmode     ::=  B[ATCH]  |  D[IALOGUE]

DEBUG  Gibt an, ob die erzeugte Phase mit dem Ada-Debugger getestet werden soll. Es können nur solche Einheiten getestet werden, die zuvor nicht mit DEBUG = NO übersetzt wurden.

= 'N'  Die gependerte Phase kann nicht mit dem Debugger getestet werden, auch wenn die Übersetzungseinheiten mit DEBUG = FULL oder DEBUG = RESTRICTED übersetzt wurden.

= RESTRICTED  Es wird nur passives Testen erlaubt, auch wenn einzelne Übersetzungseinheiten mit DEBUG = FULL übersetzt wurden.

= FULL  Der beim Übersetzen angegebene Testmodus wird von den Übersetzungseinheiten übernommen (d.h. eine Übersetzungseinheit, die mit DEBUG = FULL übersetzt wurde, kann auch aktiv getestet werden; eine Übersetzungseinheit, die mit DEBUG = RESTRICTED übersetzt wurde, kann nur passiv getestet werden; Übersetzungseinheiten mit DEBUG = NO können überhaupt nicht getestet werden).

INSTRFILE  Dieser Parameter legt den Namen der Ausgabedatei fest, in der die Anweisungen für den BS2000-Binder abgelegt werden. In Abhängigkeit vom Parameter RUNMODE = wird ent-
weder eine BS2000-DO-Prozedur oder eine BS2000-Enter-Prozedur erzeugt.

= filename
Die Binderanweisungen werden in der Datei filename abgelegt.

= (DEFAULT)
Die Binderanweisungen werden in der Datei LK.main_program abgelegt.

LISTFILE
Legt die Ausgabedatei fest, in der das Listing abgelegt werden soll.

= filename
Das Binderlisting wird in der Datei filename abgelegt.

= (DEFAULT)
Das Binderlisting wird in der Datei LK.main_program abgelegt.

LISTMODE
Legt den Umfang des erzeugten Binderlistings fest. Bei den folgenden Optionen erhält man im Binderlisting:

- ELAB: die Elaborationsreihenfolge,
- MAP: die Programmsicht des BS2000-Binders,
- OPT: die Binderoptionen,
- SORT: die alphabetisch sortierte Liste der Programmabschnitte,
- XREF: die Querverweisliste des BS2000-Binders,
- UNITS: ein Verzeichnis aller Einheiten in der Hülle.

PHASEFILE
Legt den Namen der Datei fest, in die der Binder das gebundene Programm schreiben soll. Ist bereits eine Datei mit diesem Namen vorhanden, so wird diese ersetzt.

= filename
Das gebundene Programm wird in der Datei filename abgelegt.

= DEFAULT
Das gebundene Programm wird in der Datei LK.main_program abgelegt.

RESOLVELIB
Kennzeichnet die Objektmodulbibliothek, mit der in anderen Sprachen geschriebenen Unterprogrammen.

= filename
Die Objektmodulbibliothek hat den Namen filename.

= (NONE)
Es wird keine weitere Objektmodulbibliothek verwendet.

RUNMODE
Legt fest, ob das Binden im Dialog oder als Batch-Prozeß durchgeführt wird.

= BATCH
Das Ada-Binden und das BS2000-Binden werden zusammen als Batch-Prozeß durchgeführt.

= DIALOGUE
In diesem Fall wird das Ada-Binden im Dialog durchgeführt, und die Steueranweisungen an den BS2000-Binder werden in
die BS2000-Datei LK.main._program geschrieben. Der BS2000-Binder kann anschließend im Nebenprozeß mit "/DO LK.main._program" aufgerufen werden.

XS Definiert, in welcher Betriebssystemversion das Programm ablaufen soll.

= YES Es soll in einer XS-fähigen Version ablaufen.

= NO Es soll in der 24-Bit-Version von BS2000 ablaufen.