UNCLASSIFIED

FILE COPY

2

r Date Entered)

AD-A208 498

ATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
Ada Compiler Validation Summary Report: NAVAL UNDERWATER SYSTEMS COMMAND, ADAUYK43 (ALS/N Ada/L), Version 1.0, VAX 11/785 (host) to AN/UYK-43 (target), 880719S1.09154	5. TYPE OF REPORT & PERIOD COVERED 19 July 1988 - 19 July 1989
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) National Bureau of Standards, Gaithersburg, Maryland, USA	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS National Bureau of Standards, Gaithersburg, Maryland, USA	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 19 July 1988
	13. NUMBER OF PAGES 108 p.
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) National Bureau of Standards, Gaithersburg, Maryland, USA	15. SECURITY CLASS (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 If different from Report)

UNCLASSIFIED

DTIC ELECTE MAY 25 1989 S D D

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ADAUYK43 (ALS/N Ada/L), Version 1.0, NAVAL UNDERWATER SYSTEMS COMMAND, National Bureau of Standards, VAX 11/785 under VAX/VMS, Version 4.5 (host) to AN/UYK-43 under Bare machine (target), ACVC 1.09

89 5 24 028

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AVF Control Number: NBS88VUSN525_3

Ada Compiler
VALIDATION SUMMARY REPORT:

Certificate Number: 880719S1.09154
NAVAL UNDERWATER SYSTEMS COMMAND
ADAUYK43 (ALS/N Ada/L), VERSION 1.0
VAX 11/785 HOST and AN/UYK-43 TARGET

Completion of On-Site Testing:
19 July 1988

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

QUALITY INSPECTED 2

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

Accesion For	
NTIS CRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

Ada Compiler Validation Summary Report:

Compiler Name: ADAUYK43 (ALS/N Ada/L), Version 1.0

Certificate Number: 880719S1.09154

Host:	Target:
VAX 11/785 under VAX/VMS, Version 4.5	AN/UYK-43 under Bare machine

Testing Completed 19 July 1988 Using ACVC 1.9

This report has been reviewed and is approved.

[illegible]

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899

[illegible]

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

John P. Solomond

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

Ada Compiler Validation Summary Report:

Compiler Name: ADAUYK43 (ALS/N Ada/L), Version 1.0

Certificate Number: 880719S1.09154

Host:	Target:
VAX 11/785 under VAX/VMS, Version 4.5	AN/UYK-43 under Bare machine

Testing Completed 19 July 1988 Using ACVC 1.9

This report has been reviewed and is approved.

[illegible]

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899

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

[illegible]

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

TABLE OF CONTENTS

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.

This 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 permitted by the Ada Standard. Six classes of test 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:

> To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
>
> To attempt to identify any unsupported language constructs required by the Ada Standard
>
> To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted National Bureau of Standards, under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). On-site testing was completed 19 July 1988, at Syscon Corporation, Washington, D.C.

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:

> Software Standards Validation Group
> Institute for Computer Sciences and Technology
> National Bureau of Standards
> Building 225, Room A266
> Gaithersburg, Maryland 20899

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. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines. Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide., December 1986.

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.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
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.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Language Maintenance	The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and Panel possible changes to the ANSI/MIL-STD for Ada.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	An Ada program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

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 compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. 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.

Class C tests check 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.

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 I 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.

Two library units, the package REPORT and the procedure CHECK FILE, 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 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. 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 the tests in the ACVC follow 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 validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

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

Compiler: ADAUYK43 (ALS/N Ada/L), Version 1.0

ACVC Version: 1.9

Certificate Number:		880719S1.09154
Host Computer:		
	Machine:	VAX 11/785
	Operating System:	VAX/VMS Version 4.5
	Memory Size:	16 Mb
Target Computer:		
	Machine:	AN/UYK-43
	Operating System:	Bare machine
	Memory Size:	16 Mb
Communications Network:		Portal-43

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:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See test D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined type LONG_INTEGER in the package STANDARD. (See tests B86001BC and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilatio.ı, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Expression evaluation.

  Apparently some default iritialization expressions or record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

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

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Sometimes NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_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.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

- Rounding.

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

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

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

- 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:

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR (See test C36003A.)

NUMERIC_ERROR is raised when an array type with INTEGER'LAST + 2 components is declared.) (See test C36202A.)

NUMERIC_ERROR is raised when an array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

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 no exception. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be 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.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with disciminants, the expression appears to be 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.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

Not all choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is not supported, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

Record representation clauses are not supported. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

- Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

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

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

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for DIRECT_IO.

Overwriting to a sequential file truncates the file to last element written. (See test CE2208B.)

An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and cannot be created in IN_FILE mode. (See test EE3102C.)

- Generics.

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

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

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

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tests, 28 tests had been withdrawn because of test errors. The AVF determined that 447 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing. Modifications to the code, processing, or grading for 24 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

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	107	1048	1416	17	13	46	2647
Inapplicable	3	3	437	0	4	0	447
Withdrawn	3	2	21	0	2	0	28
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	181	449	466	245	165	98	141	327	137	36	234	3	165	2647
Inapplicable	23	123	208	3	0	0	2	0	0	0	0	0	88	447
Withdrawn	2	14	3	0	1	1	2	0	0	0	2	1	2	28
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 28 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C	C35904A
C35904B	C35A03E	C35A03R	C37213H	C37213J	C37215C
C37215E	C37215G	C37215H	C38102C	C41402A	C45332A
C45614C	E66001D	A74106C	C85018B	C87B04B	CC1311B
BC3105A	AD1A01A	CE2401H	CE3208A		

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, 447 test were inapplicable for the reasons indicated:

C35508I..J (2 tests) and C35508M..N (2 tests) use enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1). These clauses are not supported by this compiler.

C35702A uses SHORT_FLOAT which is not supported by this implementation.

C35702B uses LONG_FLOAAT which is not supported by this implementation.

A39005G uses a record representation clause that is not supported by this compiler.

The following (14) tests use SHORT_INTEGER, which is not supported by this compiler.

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	

C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.

C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.

C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this compiler.

B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

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

The following 79 tests are inapplicable because this implementation supports only tape or terminal IO.

CE2102B	CE2102F..G(2)	CE2102I..K(3)	CE2103B
CE2104C..D(2)	CE2105B	CE2106B	CE2107A..I(9)
CE2108A..D(4)	CE2109B	CE2110B..C(2)	CE2111B
CE2111D..E(2)	CE2111G..H(2)	CE2115A	CE2204A..B(2)
CE2401A..C(3)	CE2401E..F(2)	CE2402A	CE2404A
CE2405B	CE2406A	CE2407A	CE2408A
CE2409A	CE2410A	CE2411A	CE3104A
CE3109A	CE3111A..E(5)	CE3112A..B(2)	CE3114B

CE3115A	CE3203A	CE3402B	CE3404A
CE3408B	CE3411C	CE3412C	CE3413C
CE3602C..D(2)	CE3605C	CE3605E	CE3704A
CE3804C	CE3804I	CE3805A..B(2)	CE3806A
CE3806D	CE3905A	CE3906A	

CE2102C uses a string which is illegal as an external file name for SEQUENTIAL_IO. No illegal file names exist.

CE3102B expects exception to be raised because of an illegal file name. This implementation does not restrict file names.

The following 327 tests require a floating-point accuracy that exceeds the maximum of 6 digits supported by this implementation:

C24113C..Y (23 tests)	C35705C..Y (23 tests)
C35706C..Y (23 tests)	C35707C..Y (23 tests)
C35708C..Y (23 tests)	C35802C..Z (24 tests)
C45241C..Y (23 tests)	C45321C..Y (23 tests)
C45421C..Y (23 tests)	C45521C..Z (24 tests)
C45524C..Z (24 tests)	C45621C..Z (24 tests)
C45641C..Y (23 tests)	C46012C..Z (24 tests)

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 sub-tests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

C4A012B raises NUMERIC_ERROR rather than CONSTRAINT_ERROR. The test has been evaluated and graded as PASSED.

Modifications were required for 24 Class B tests.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B2A003A	B2A003B	B2A003C	B33201C	B33202C
B33203C	B33301A	B37106A	B37201A	B37301I
B37307B	B38001C	B38003A	B38003B	B38009A
B38009B	B44001A	B51001A	B54A01C	B54A01L
B95063A	BC1008A	BC1201L	BC3013A	

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the ADAUYK43 (ALS/N Ada/L) 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 ADAUYK43 (ALS/N Ada/L) using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 11/785 operating under VAX/VMS, Version 4.5 host and a AN/UYK-43 target operating under Bare machine. The host and target computers were linked via Portal-43.

A magnetic tape containing all tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on-site after the magnetic tape was loaded. Tests requiring modifications during the prevalidation testing were not 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 and linked on the VAX 11/785, and all executable tests were run on the AN/UYK-43. Object files were linked on the host computer, and executable images were transferred to the target computer via Portal-43. Results were printed from the host computer, with results being transferred to the host computer via Portal-43.

The compiler was tested using command scripts provided by Control Data Corporation and reviewed by the validation team. The compiler was tested using all default option settings without exception.

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target 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 Syscon Corporation, Washington, D.C., and was completed on 19 July 1988.

APPENDIX A

CONFORMANCE STATEMENT

DECLARATION OF CONFORMANCE

Compiler Implementor: Control Data Corporation, Software Programs Division
Ada[R] Validation Facility: ASD/SCOL, Wright-Patterson AFB, OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

Base Compiler Name:	ADAUYK43 (ALS/N Ada/L)	Version:	0.5
Host Architecture ISA:	Digital VAX	OS&VER #:	VMS 4.5
Target Architecture ISA:	AN/UYK-43	OS&VER #:	N/A

Implementor's Declaration

I, the undersigned, representing Control Data Corporation, Software Programs Division, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that Naval Sea Systems Command, Department of the Navy, is the owner of record of the Ada Language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for the Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

[illegible] Date: April 5 1988
D. L. Whitt, Program Manager

Owner's Declaration:

I, the undersigned, representing Naval Sea Systems Command, Department of the Navy, take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. 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.

[illegible] Date: April 14, 1988

[R] Ada is a registered trademark of the United States Government (Ada Joint Program Office).

APPENDIX B

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 the VAX 11/785, Version 4.5, are described in the following sections which discuss topics in Appendix F of the Ada Standard. Implementation- specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is


     type INTEGER is range -2_147_483_647 .. 2_147_483_647

     type LONG_INTEGER is range -9_223_372_036_854_775_807 ..
                                 9_223_372_036_854_775_807;

     type FLOAT is digits 6 range -(16#0.FF_FFFF#E63)..
                                   (16#0.FF_FFFF#E63);

     type DURATION is delta 2.0**(-14) range -131_071.0 ..
                                              131_071.0;


end STANDARD;
```

APPENDIX F

APPENDIX F OF THE ADA LRM FOR THE ADAUYK43 TOOLSET.

IMPLEMENTATION-DEFINED PRAGMAS

pragma DEBUG;

DEBUG Applies to the entire compilation unit in which the pragma appears.

This pragma enables one or more debugging features. These debugging features shall be sufficient to support the requirements of the Embedded Target Debugger and the Run-Time Debugger.

pragma EXECUTIVE [(arg)];

EXECUTIVE Applies to the library unit in which the pragma appears, and to any corresponding secondary units.

This pragma shall provide user-written Ada programs and RTE functions access to the machine-dependent facilities of the embedded target computer from the Ada language implemented by the Ada/L Code Generator. Program units compiled with the EXECUTIVE pragma shall have:

Direct access to all the services of the Run-Time Executive, Run-Time Support Library, and Run-Time Loader that are available to the RTE Functions (i.e., all interfaces specified in the Ada/L Interface Specification). The only access to the Run-time Executive is through the RTEXEC_GETEWAY;

The ability to execute the "privileged" instructions of the embedded target computer (these instructions shall be checked for and flagged as a WARNING if they occur in program units compiled with the NO_EXECUTIVE option or without the EXECUTIVE pragma);

The ability to specify address clauses for hardware interrupts of the embedded target computers (these address clauses shall be checked for and flagged as a WARNING if they occur in program units compiled with the NO_EXECUTIVE option or without the EXECUTIVE pragma);

The ability to execute in the "executive" state(s) of the embedded target computer (program units compiled with the NO_EXECUTIVE option or without the EXECUTIVE pragma shall be limited to the "task" state);

The ability to call and be called by other program units compiled with the NO_EXECUTIVE option or without the EXECUTIVE pragma (the state will be changed as appropriate);

Continued use of the STATIC, UNMAPPED, DEBUG, and MEASURE pragmas.

The EXECUTIVE pragma has an optional argument which is PRIVILEGE. The use of the PRIVILEGE argument directs the compiler to generate privileged instructions where possible.

pragma MEASURE (extraction_ set, [arg,...]);

MEASURE   No scope is associated with MEASURE.

This pragma enables one or more performance measurement features, including the specification of objects in extraction sets. These performance measurement features shall be sufficient to support the requirements of the Run-Time Performance Measurement Aids.

pragma STATIC;

STATIC   Applies to the library unit in which the pragma appears, and to any corresponding secondary units.

The pragma STATIC is only allowed immediately after the declaration of a task body containing an immediate interrupt entry. The effect of this pragma will be to allow generation of nonreentrant and nonrecursive code in a compilation unit, and to allow static allocation of all data in a compilation unit. This pragma shall be used to allow for procedures within immediate (fast) interrupt entries. The effect will be for the compiler to generate nonreentrant code for the affected procedure bodies. If a STATIC procedure is called recursively, the program is erroneous.

pragma TITLE (arg);

TITLE   Applies to the compilation unit in which the pragma appears.

This is a listing control pragma. It takes a single argument of type string. The string specified will appear on the second line of each page of every listing produced for the compilation unit. At most one such pragma may appear for any compilation unit, and it must be the first lexical unit in the compilation unit (excluding comments).

pragma TRIVIAL_ENTRY (NAME: entry_simple_name);

TRIVIAL_ENTRY   Applies only to the entry named in its argument.

This pragma is only allowed within a task specification after an entry declaration and identifies a Trivial_Entry to the system.

pragma UNMAPPED (arg [arg,...]);

UNMAPPED Applies to package data in the compilation in which the pragma appears.

The effect of this pragma is for unmapped (i.e., not consistently mapped) allocation of package data in a compilation unit. The arguments of this package are access types and variables to be unmapped. The compiler shall be free to generate a compilation unit with package data larger than the maximum allowable phase size, but not larger than the physical memory.
Information about phase sizes and memory mapping may be found in the Run-Time Environment Handbook.

LANGUAGE-DEFINED PRAGMAS

This paragraph specifies implementation specific changes to those pragmas described in Appendix B of ANSI/MIL-STD-1815A.

pragma CONTROLLED (arg);
CONTROLLED Applies only to the access type names in its argument.

No Change.

pragma ELABORATE (arg [arg,...]);

ELABORATE Applies to the entire compilation unit in which the pragma appears.

No Change.

pragma INLINE (arg [arg,...]);

INLINE Applies only to subprogram names in its arguments. If the argument is an overloaded subprogram name, the INLINE pragma applies to all definitions of that subprogram name which appear in the same declarative part as the INLINE pragma.

Subprograms specified as an argument to an INLINE pragma, and which are directly recursive, are not expanded in-line at the point of the recursive invocation. Such calls use normal Ada subprogram calling semantics.

pragma INTERFACE (arg, arg);

INTERFACE Applies to all invocations of the named imported subprogram.

The first argument specifies the language and type of interface to be used in calls used to the externally supplied subprogram, specified by the second argument. The allowed values for the first argument (language name) are MACRO_NORMAL and MACRO_QUICK. MACRO_NORMAL indicates that parameters will be passed on the stack and the calling conventions used for normal Ada subprogram calls (see Section 3.4.14.2 of Ada/L_Intf_Spec]) will apply. MACRO_QUICK is used in RTLIB routines to indicate that parameters are passed in registers.

The user must ensure that an assembly-language body container will exist in the library before linking.

pragma LIST (arg);

LIST Applies from the point of its appearance until the next LIST pragma in the source or included text, or if none, the end of the compilation unit.

No Change.

pragma MEMORY_SIZE (arg);

MEMORY_SIZE   Applies to the entire Program Library in which the pragma appears.

This pragma must appear at the start of the first compilation when creating a program library. If it appears elsewhere, a diagnostic of severity WARNING is generated and the pragma has no effect.

pragma OPTIMIZE (arg);

OPTIMIZE   Applies to the entire compilation unit in which the pragma appears.

The argument is either TIME or SPACE. The default is SPACE. This pragma will be effective only when the OPTIMIZE option has been given to the compiler, as described in Appendix 20 of ALS/N_Spec,

pragma PACK (arg);

PACK   Applies only to the array or record named as the argument.

No Change.

pragma PAGE

PAGE   No scope is associated with PAGE.

No Change.

pragma PRIORITY (arg);

PRIORITY   Applies to the task specification in which it appears, or to the environment task if it appears in the main subprogram.

The argument is an integer static expression in the range 0..15, where 0 is the lowest user-specifiable task priority and 15 is the highest. If the value of the argument is out of range, the pragma will have no effect other than to generate a WARNING diagnostic. A value of zero will be used if priority is not defined. The pragma will have no effect when not specified in a task (type) specification or the outermost declarative part of a subprogram, it will have no effect unless that subprogram is designated as the main subprogram at link time.

pragma SHARED (arg);

SHARED   Applies to the scope of the scalar or access variable named by the argument.

No change.

pragma STORAGE_UNIT (arg);

STORAGE_UNIT Applies to the entire Program Library in which the pragma appears.

This pragma must appear at the start of the first compilation when creating a program library. If it appears elsewhere, a diagnostic of severity WARNING is generated and the pragma has no effect.

pragma SUPPRESS (arg[,arg]);

This pragma is unchanged with the following exceptions:

Suppression of OVERFLOW_CHECK applies only to integer operations; and a SUPPRESS pragma has effect only within the compilation unit in which it appears except that suppression of ELABORATION_CHECK applied at the declaration of a subprogram or task unit applies to all calls or activations.

pragma SYSTEM_NAME (arg);

This pragma must appear at the start of the first compilation when creating a program library. If it appears elsewhere, a diagnostic of severity WARNING is generated and the pragma has no effect. The only allowable value for the argument is the enumeration literal AN/UYK-43. For other values a WARNING diagnostic is generated.

IMPLEMENTATION-DEFINED ATTRIBUTES

There are two implementation-defined attributes in addition to the predefined attributes found in Appendix A of ANSI/MIL-STD-1815A. These are defined below.

p'PTI_ID

Yields a value of type intentry.pti_entry. The prefix of this attribute identifies a fully qualified interrupt entry. This attribute is used to pass an entry name to a procedure.

p'PHYSICAL_ADDRESS   For a prefix p that denotes a data object.

Yields a value of type system.physical_address, which corresponds to the absolute address in physical memory of the object named by p. This attribute is used to support operations associated with the pragma UNMAPPED.

The following definitions augment the language-required definitions of the predefined attributes found in Appendix A of ANSI/MIL-STD01815A.

T'MACHINE_ROUNDS	is false.
T'MACHINE_RADIX	is 16.
T'MACHINE_MANTISSA	is 6.
T'MACHINE_EMAX	is 63.
T'MACHINE_EMIN	is -64.
T'MACHINE_OVERFLOWS	is true.

APPENDIX F OF THE ADA LRM FOR THE ADAUYK43 TOOLSET.

The package SYSTEM is as follows:

```
--      c 1987 United States Government as represented by
--        the Secretary of Navy. ALL RIGHTS RESERVED.
--
--        (The U.S. Government possesses the unlimited rights
--        throughout the world for Government purposes to
--        publish, translate, reproduce, deliver, perform, and
--        dispose of the technical data, computer software, or
--        computer firmware contained herein; and to authorize
--        others to do so.)

-- REVISION HISTORY:
--    23 FEB 1988 DET
--       ISTR 246 :
--       o Added definitions for _CHK type exceptions that were
--         present in the ADAVAX system but missing from ADA/L
--    25 JAN 1988 ROS
--       ISCP 236 :
--       o Added interrupt address constants and FUNCTION address_of.
--    10 Mar 1987 TCJ
--       Coded from PDL

PACKAGE SYSTEM IS

--| JUSTIFICATION:
--|
--|     SYSTEM contains the definitions of certain
--| configuration-dependent characteristics (see Section 13.7 of
--| [ANSI/MIL-STD-1815A]) of Ada/L(43).
--|
--|     SYSTEM also provides system dependent logical routines
--| and conversion routines.
--|
--| Assumptions:
--|
--|     SYSTEM is targeted for Ada/L(43).

--| TYPES and DATA:
--|
--|     See below.

           TYPE name IS (anuyk43);
                        -- only one compatible system name.

           system_name : CONSTANT system.name := system.anuyk43;
                        -- name of current system.

           storage_unit : CONSTANT := 32;
                         -- word-oriented system (not configurable)

           memory_size : CONSTANT := 1_048_576;
                        -- 2**20
                        -- virtual memory size (not configurable).
```

```
TYPE address IS RANGE 0..system.memory_size - 1;
              -- virtual address.

-- FOR address'SIZE USE 32;
                 -- virtual address is a 32-bit quantity.

null_addr : CONSTANT address := 0;
            -- Indicates a NULL address.

--
-- Address clause (interrupt) addresses
--
Intercomputer_Timeout_address : CONSTANT address := 11;
Confidence_Test_Fault_address : CONSTANT address := 12;
Data_Pattern_Breakpoint_address : CONSTANT address := 20;
Operand_Breakpoint_Match_address : CONSTANT address := 21;
DCU_Status_Interrupt_address : CONSTANT address := 23;
Instruction_Breakpoint_Match_address : CONSTANT
   address := 27;
RPD_Underflow_address : CONSTANT address := 28;
IOC_Confidence_Test_Fault_address : CONSTANT
   address := 37;
IOC_Breakpoint_Match_address : CONSTANT address := 38;
-- Pseudo-Interrupts
System_Damage_address : CONSTANT address := 41;
Program_Damage_address : CONSTANT address := 42;
-- I/O Interrupts
--    User should program
--       FOR entry-name USE AT system.address_of(
--          interrupt=>interrupt_name, for_channel=>
--              channel_number);
--    e.g.
--       USE system;
--       ..
--       FOR el USE AT address_of(ioc_cp_interrupt,
--                     for_channel=>5);
--
--    (Declaration of FUNCTION address_of is found below)
--
IOC_CP_Interrupt : CONSTANT integer := 1000;
IOC_External_Interrupt_Monitor : CONSTANT integer := 1001;
IOC_External_Function_Monitor : CONSTANT integer := 1002;
IOC_Output_Data_Monitor : CONSTANT integer := 1003;
IOC_Input_Data_Monitor : CONSTANT integer := 1004;
SUBTYPE IO_interrupts IS
   INTEGER RANGE IOC_CP_Interrupt..IOC_Input_Data_Monitor;
SUBTYPE channel_numbers IS INTEGER RANGE 0..63;

physical_memory_size : CONSTANT := 2**31;
                       -- maximum physical memory size
                          -- (not configurable)

TYPE physical_address IS
   RANGE 0..system.physical_memory_size - 1 ;
   -- absolute address.
```

```
null_phys_addr : CONSTANT physical_address := 0;
                 -- Indicates a NULL physical address.

TYPE word IS NEW INTEGER;

     -- objects of this type occupy one target_computer
     -- word *32 bits on the AN/UYK-43).
     -- UNCHECKED_CONVERSION must be used to interpret
     -- the value for an object of this type from Ada.


min_int : CONSTANT := -((2**63)-1);
          -- most negative integer.

max_int : CONSTANT := (2**63)-1;
          -- most positive integer.

max_digits : CONSTANT := 15;
   -- most decimal digits in floating point constraint.

max_mantissa : CONSTANT := 31;
   -- most binary digits for fixed point subtype.


fine_delta : CONSTANT
   := 2#0.0000_0000_0000_0000_0000_0000_0000_001#;
   -- 2**(-31) is minimum fixed point constraint.

tick : CONSTANT := 4.8828125e-05;
       -- 1/20480 seconds is the basic clock period.

SUBTYPE priority IS integer RANGE 0..15;
                 -- task priority, lowest = default = 0.

TYPE entry_kind IS (normal, immediate);
                -- enumeration type for use with PRAGMA
                -- INTERRUPT_HANDLER_TASK.

-- The following exceptions are provided as a "convention"
-- whereby the Ada program can be compiled with all
-- implicit checks suppressed (i.e. PRAGMA SUPPRESS or
-- equivalent) and explicit checks included as necessary
-- that RAISE the appropriate exception when required.
-- The explicitly raised execption is either handled or
-- the Ada program terminates.

ACCESS_CHECK          : EXCEPTION;
DISCRIMINANT_CHECK    : EXCEPTION;
INDEX_CHECK           : EXCEPTION;
LENGTH_CHECK          : EXCEPTION;
RANGE_CHECK           : EXCEPTION;
DIVISION_CHECK        : EXCEPTION;
OVERFLOW_CHECK        : EXCEPTION;
ELABORATION_CHECK     : EXCEPTION;
STORAGE_CHECK         : EXCEPTION;
```

```
          -- implementation-defined exceptions.

          UNRESOLVED_REFERENCE : EXCEPTION;
          SYSTEM_ERROR         : EXCEPTION;
          CAPACITY_ERROR       : EXCEPTION;
          -- The exception CAPACITY_ERROR is raised by the RTExec
          -- when Pre-RunTime specified resouce limits are exceeded.

--| CREATED TASKS:
--|
--|     None.


--| SUBPROGRAMS AND TASKS:
--|
--|

    FUNCTION ADDRESS_OF
    -- returns the system.address of the given Class III interrupt
    -- for the specified channel
       (interrupt : IN IO_interrupts;
                    -- The name of the interrupt,

        for_channel : IN channel_numbers
                    -- The channel number.

       ) RETURN address;
                -- The address to be used in the representation
                -- (address) clause.

    PRAGMA INTERFACE (MACRO_NORMAL,ADDRESS_OF);


    FUNCTION "AND"
    -- returns the logical 32 bit 'AND' between two integers.
       (operand_a : IN integer;
                    -- The first operand.

        operand_b : IN integer
                    -- The second operand

       ) RETURN integer;
                    -- The results.

    PRAGMA INTERFACE (MACRO_NORMAL, "AND");

    FUNCTION "NOT"
    -- returns the logical 32 bit 'NOT' of an integer.
       (operand_a : IN integer
                    -- The first operand.

       ) RETURN integer;
                    -- The results.

    PRAGMA INTERFACE (MACRO_NORMAL, "NOT");
```

```
   FUNCTION "OR"
   -- returns the logical 32 bit 'OR' between two integers.
      (operand_a : IN integer;
                   -- The first operand.

       operand_b : IN integer
                   -- The second operand

      ) RETURN integer;
                   -- The results.

   PRAGMA INTERFACE (MACRO_NORMAL, "OR");

END SYSTEM;
```

REPRESENTATION AND DECLARATION RESTRICTIONS

Representation specifications are described in Section 13 of ANSI/MIL-STD-1815A. Declarations are described in Section 3 of ANSI/MIL-STD-1815A.

In the following specifications, the capitalized word SIZE indicates the number of bits used to represent an object of the type under discussion. The upper case symbols D, L, R, correspond to those discussed in Section 3.5.9 of ANSI/MIL-STD-1815A.

Enumeration Types

In the absence of a representation specification for an enumeration type "t", the internal representation of t'FIRST is 0. The default size for a stand-alone object of enumeration type "t" is 32, so the internal representations of t'FIRST and t'LAST both fall within the range -(2**15 - 1) .. 2**15 - 1.

Length specifications of the form

```
FOR t'SIZE USE n;
```

and/or enumeration representations of the form

```
FOR t USE aggregate;
```

are permitted for n in 2..32, provided the representations and the SIZE conform to the relationship specified above, or else for n in 1..32, provided that the internal representation of t'FIRST >= 0 and the representation of t'LAST <= 2**(t'SIZE) - 1.

For components of enumeration types within packed composite objects, the smaller of the default stand-alone SIZE or the SIZE length specification is used.

Enumeration representations for types derived from the predefined type standard.boolean will not be accepted, but length specifications will be accepted.

Arrays and Records

A length specification of the form

```
FOR t'size USE n;
```

may cause arrays and records to be packed, if required, to accommodate the length specification. If the size specified is not large enough to contain any value of the type, a diagnostic message of severity ERROR is generated.

The PACK pragma may be used to minimize wasted space between components of arrays and records. The pragma causes the type representation to be chosen such that the storage space requirements are minimized at the possible expense of data access time and code space.

A record type representation specification may be used to describe the allocation of components in a record. Bits are numbered 0..31 from the right. Bit 32 starts at the right of the next higher numbered word. Each location specification must allow at least n bits of range, where n is large enough to hold any value of the subtype of the component being allocated. Otherwise, a diagnostic message of severity ERROR is generated. Components that are arrays, records, tasks, or access variables may not be allocated to specified locations. If a specification of this form is entered, a diagnostic message of severity ERROR is generated.

The alignment clause of the form

```
AT MOD n
```

may only specify alignments of 1 or 2 (corresponding to word or doubleword alignment, respectively).

If it is determinable at compile time that the SIZE of a record or array type or subtype is outside the range of standard.integer, a diagnostic of severity WARNING is generated. Declaration of such a type or subtype would raise NUMERIC_ERROR when elaborated.

Address Clauses

Refer to Section 13.5 of ANSI/MIL-STD-1815A for a description of address clauses. All rules and restrictions described there apply. In addition, the following restrictions apply.

An address clause designates a single task entry only. The appearance of a data object, subprogram, package, or task unit name in an address clause is not allowed, and will result in the generation of a diagnostic of severity ERROR.

An address clause may designate a single task entry. Such an address clause is allowed only within a task specification compiled with the EXECUTIVE compiler option. The meaningful values of the simple_expression are the allowable interrupt entry addresses as defined in Appendix G of RTOS_PPS. The use of other values will result in the raising of a PROGRAM_ERROR exception upon creation of the task.

If more than one task entry is equated to the same interrupt entry address, the most recently executed interrupt entry registration permanently overrides any previous registrations.

At most one address clause is allowed for a single task entry. Specification of more than one interrupt address for a task entry is erroneous.

SYSTEM GENERATED NAMES

There are no system generated names.

ADDRESS CLAUSES

Refer to Section 13.5 of ANSI/MIL-STD-1815A for a description of address clauses. All rules and restrictions described there apply. In addition, the following restrictions apply.

An address clause designates a single task entry only. The appearance of a data object, subprogram, package, or task unit name in an address clause is not allowed, and will result in the generation of a diagnostic of severity ERROR.

An address clause may designate a single task entry. Such an address clause is allowed only within a task specification compiled with the EXECUTIVE compiler option. The meaningful values of the simple_expression are the allowable interrupt entry addresses as defined in Appendix G of RTOS_PPS. The use of other values will result in the raising of a PROGRAM_ERROR exception upon creation of the task.

If more than one task entry is equated to the same interrupt entry address, the most recently executed interrupt entry registration permanently overrides any previous registrations.

At most one address clause is allowed for a single task entry. Specification of more than one interrupt address for a task entry is erroneous.

UNCHECKED_CONVERSION

Refer to Section 13.10.2 of ANSI/MIL-STD-1815A for a description of UNCHECKED_CONVERSION. It is erroneous if the User_Written_Ada_Program performs UNCHECKED_CONVERSION when the source and target objects have different sizes.

INPUT/OUTPUT

I/O in Ada/L is performed solely on external files. No allowance is provided in the I/O subsystem for memory resident files (i.e., files which do not reside on a peripheral device). This is true even in the case of temporary files. With the external files residing on the peripheral devices, Ada/L makes the further restriction that only one file may be open on an individual RD358 tape unit at a time.

The naming conventions for external files in Ada/L are of particular importance to the user. All of the system-dependent information needed by the I/O subsystem about an external file is contained in the file name. External files may be named using one of three file naming conventions: standard, temporary, and user-derived.

Standard File Names:

The standard external file naming convention used in Ada/L identifies the specific location of the external file in terms of the physical device on which it is stored. For this reason, the user should be aware of the configuration of the peripheral devices on the AN/UYK-43 at a particular user site.

Standard file names may be six to twenty characters long; however, the first six characters follow a predefined format. The first and second characters must be either "DK", "MT", or "TT", designating an AN/UYH-3(V) Recorder-Reproducer Set Magnetic Disk, the RD-358 Magnetic Tape Subsystem, or the AN/USQ-69 Data Terminal Set, respectively.

The third and fourth characters specify the channel on which the peripheral device is connected. Since there are sixty-four channels on the AN/UYK-43, the values for the third and fourth positions must lie in the range "00" to "63".

The range of values for the fifth position in the external file name (the unit number) depends upon the device specified by the characters in the first and second positions of the external file name. If the specified peripheral device is the AN/UYH-3 magnetic disk drive, then the character in the fifth position must be one of the characters "0", "1", "2", or "3". This value determines which of the four disk units available on the AN/UYH-3 is to be accessed. If the specified peripheral device is the RD-358 magnetic tape drive, the character in the fifth position must be one of the characters "0", "1", "2", or "3". This value determines which of the four tape units available on the RD-358 is to be accessed. If the specified peripheral device is the AN/USQ-69 militarized display terminal, the character in the fifth position must be a "0". This is the only allowable value for the unit number when the AN/USQ-69 is connected to a paralled I/O channel. This is because the AN/USQ-69 may have only one unit on a parallel channel.

The colon, ":", is the only character allowed in the sixth position. If any character other than the colon is in this position, the file name will be considered non-standard and the file will reside on the default device defined during the elaboration of CONFIGURE_IO.

Positions seven through twenty are optional to the user-written Ada program and may be used as desired. These positions may contain any printable character the user chooses in order to make the file name more intelligible. Embedded blanks, however, are not allowed.

The location of an external file on a peripheral device is thus a function of the first six characters of the file name regardless of the characters that might follow. For example, if the external file "MT000:Old_Data" has been created and not subsequently closed, an attempt to create the external file "MT000:New_Data" will cause the exception DEVICE_ERROR (rather than NAME_ERROR or USE_ERROR) to be raised because the peripheral device on channel "00" and cartridge "0" is already in use.

The user is advised that any file name beginning with "xxxxx:" (where x denotes any printable character) is assumed to be a standard external file name. If this external file name does not conform to the Ada/L standard file naming conventions, the exception NAME_ERROR will be raised.

Temporary File Names:

Section 14.2.1 of [ANSI/MIL-STD-1815A] defines a temporary file to be an external file that is not accessible after completion of the main subprogram. If the null string is supplied for the external file name, then the external file is considered temporary. In this case, the high level I/O packages internally create an external file name to be used by the lower level I/O packages. The internal naming scheme used by the I/O subsystem is a function of the type of file to be created (text, direct or sequential), the temporary nature of the external file, and the number of requests made thus far for creating temporary external files of the given type. This scheme is consistent with the requirement specified in [ANSI/MIL-STD-1815A] that all external file names be unique.

The first three characters of the file name are "TEX", "DIR", or "SEQ". The next six characters are "_TEMP_". The remaining characters are the image of an integer which denotes the number of temporary files of the given type successfully created. There are two types of temporary files; one is used by SEQUENTIAL_IO and DIRECT_IO, and the other is used by TEXT_IO.

For instance, the temporary external file name "TEX_TEMP_10" would be the name of the tenth temporary external file successfully created by the user-written Ada program through calls to TEXT_IO.

User-Derived File Names:

A random string containing a sequence of characters of length one to twenty may also be used to name an external file. External files with names of this nature are considered to be permanent external files. The user is cautioned to refrain from using names which conform to the scheme used by the I/O subsystem to name temporary external files (see subsection (b) above).

Ada/L restricts the creation of files to those of mode "out_file."

In the case of the AN/USQ-69, where one file of mode "in_file" and one file of mode "out_file" may be open on the same terminal, the user must open two separate files in order to read from and write to the terminal.

If the peripheral device is an RD-358, the file is assumed to be an Ada/L compatible file, including the file information header block unless a foreign tape format is specified by a form parameter with a NOHEAD option. If the underlying I/O system is expecting an Ada/L compatible header record and none is found, the tape is rewound and the exception DATA_ERROR is raised. If the peripheral device is an RD-358 and the tape format is foreign, the file must be opened with the form parameter specifying the NOHEAD option. This suppresses the check for Ada/L compatibility and assumes that no header record exists for the file.

Failing to close a file on a terminal has no adverse effects.

If the user fails to close a file on the magnetic tape prior to the end of the main subprogram, the tape may not be positioned correctly for future tape operations.

Issuing a CREATE or OPEN with the form option NOREWIND after loading the tape and manually placing the tape in an arbitrary position will also give unpredictable results.

Failure to close a file on a magnetic disk prior to the end of the main subprogram may result in loss of date. Since the disk I/O is buffered, data may reside in local buffers which could be lost if the buffers are not flushed prior to the completion of the main subprogram.

I/O Arguments:

The I/O arguments allow a user access to the features of the RD-358/UYK tape drive. Specifically the arguments control the positioning and formatting of the tape prior to I/O operations to the tape. This section describes the appropriate use of these I/O arguments and the consequences of inappropriate use.

The RD-358/UYK tape drive has been implemented to allow several different formatting/positioning options. The user selects the arguments for a particular file and conveys them to the I/O System via the FORM parameter.

The FORM parameter is a string literal of which a maximum of twenty characters is processed. If the supplied form string is longer than the maximum allowed form string (20 characters), the exception USER_ERROR will be raised. The string literal is interpreted as a sequence of arguments. If the user wishes to utilize the default arguments, a FORM parameter need not be supplied.

Only the first two arguments within the string are processed. All following characters or arguments will cause the USE_ERROR to be raised. The arguments are not case sensitive. The arguments must be separated by at least one delimiter. A legal delimiter consists of a comma or blank. Extra delimiters are ignored. Of the recognized arguments, at most one formatting and one positioning arguments are allowed. If conflicting arguments are used, the exception USE_ERROR will be raised.

The two procedures which use the form parameter are CREATE and OPEN. For the CREATE procedure any of the possible arguments may be applied. The APPEND argument (discussed below) is not legal for the OPEN procedure; if used with an OPEN procedure a USE_ERROR exception will be raised.

Positioning arguments allow control of tape before its use. The following positioning arguments are available to the user:

a. REWIND - specifies that a rewind will be performed prior to the requested operation.

b. NOREWIND - specifies that the tape remains positioned as is.

c. APPEND - specifies that the tape be positioned at the logical end of tape (LEOT) prior to the requested operation. The LEOT is denoted by two consecutive tape_marks.

The formatting argument specifies information about tape format. If a formatting argument is not supplied, the file is assumed to contain a format header record determined by the ALS/N I/O system. The following formatting argument is available to the user:

a. NOHEAD - specifies that the designated file has no header record. This argument allows the reading and writing of tapes used on computer systems using different header formats.

Although I/O arguments affect the CREATE and OPEN procedures in a similiar manner, each procedure has unique situations that it handles. These distinctive characteristics of the CREATE procedures are as follows:

a. If a file is created with a REWIND argument, the tape rewinds to the beginning of tape before the CREATE operation takes place. No information following the current file will be accessible to the user.

b. If a file is created with the NOREWIND argument, the file is created with the tape positioned as is. No information following the current file will be accessible to the user. If a file is created with no positioning argument, the default is NOREWIND.

c. If a file is created with argument APPEND, the tape is forwarded to the LEOT before the CREATE operation takes place.

d. If an attempt to create a file with argument APPEND on a blank tape is made, a DEVICE_ERROR exception will be raised.

e. If a file is created with argument NOHEAD, the writing of a header record to tape is suppressed. All future references to the file must be done via positioning of the tape, not by name.

f. Use of the positioning argument may allow multiple files with the same name to be created on the tape. Invoking the CREATE procedure will not cause a search for an existing file of the same name.

A description of the OPEN procedure's distinctive characteristics are as follows:

a. If a file is opened with mode out_file, the user is allowed to write to the file. But, all data in the current file and all data in succeeding files is lost if there is an actual write to the opened file (i.e. the data is not lost as a result of the OPEN request but as a result of a WRITE request). The LEOT is written after each write to tape, thus causing all data following the LEOT tape mark to be inaccessible.

b. If a file is opened with argument REWIND, and the file contains a header record, the tape is rewound to the beginning of tape before searching for the specified file. The first file with the specified name is opened. If a file is opened with no positioning argument, the default is REWIND.

c. If a file is opened with argument NOREWIND and the file contains a header record, the tape remains positioned as is. The searching for the specified file begins from the current position of the tape. The first file with the specified name is opened. If no file of the specified name exists after the current position, the exception NAME_ERROR will be raised.

d. If a file is opened with arguments REWIND and NOHEAD, the tape is rewound to the beginning of tape and the first file on the tape is opened.

e. If a file is opened with argument NOREWIND and NOHEAD, the file at which the tape is currently positioned is opened.

f. If a file is opened with the default arguments (i.e. the file contains a header record and is rewound to the beginnng of tape) the first file with the specified name is opened.

Other distinctive characteristics of the Tape I/O subsystem are as follows:

a. If NOHEAD argument is specified it is assumed that NONE of the files on the tape has a file header. By default if the NOHEAD argument has not been specified all files on the tape are assumed to contain file headers. If a mixture of files with headers and without headers occur on the same tape, results are unpredictable.

b. The CLOSE procedure positions the tape at the start of the following file, if one exists, or else at the LEOT.

c. The results of the DELETE procedure are affected by the formatting arguments. If the NOHEAD argument was specified when the file was opened or created, then the LEOT is written. Any files following the file being deleted are also deleted and are no longer accessible to the user. If the file does contain a header record the file is marked as deleted and is no longer accessible to the user. No other files are affected.

d. The low tape sensor is treated as physical end of tape. No reading OR writing is permitted beyond this point.

The maximum permissible length of an enumeration value is the number of characters that will fit on a single line or 251, which ever is smaller.

The user, if choosing to perform I/O with an unbounded line length, should be mindful that the size of the internal text buffers is limited to 1024 characters. Successive calls to TEXT_IO.PUT can be made so long as the cumulative number of characters passed as arguments does not exceed the buffer size. If the buffer size is exceeded, the exception USE_ERROR is raised. A call to TEXT_IO.NEW_LINE or TEXT_IO.PUT_LINE empties the buffer by requesting that the low-level portion of the I/O subsystem write the contents of the buffer to the external file. The user must remember to count all of the characters already in the text buffer in addition to those passed in the argument to TEXT_IO.PUT_LINE in determining whether or not the size of the text buffer will be exceeded. If the user is performing I/O on files with a bounded line length, TEXT_IO monitors the buffer length automatically, writing the contents of the buffer to the external file whenever the length of the buffer reaches the limit specified by a prior call to TEXT_IO.SET_LINE_LENGTH.

An area of special concern to the user is the reading of complex (i.e., composite) data types through calls to an instantiation of SEQUENTIAL_IO.READ. [ANSI/MIL-STD-1815A] permits an implementation not to raise the exception DATA_ERROR during input operations from sequential files for which the data type is complex. Ada/L does not support the checking of data that is read from a sequential file against the instantiated element type.

Ada/L tape device drivers limit their support to peripheral equipment in the following ways: no more than one RD-358 device per channel is allowed; no more than one open file per tape and no more than four tapes per device is permitted.

The Ada/L terminal device driver limits its support to peripheral equipment in the following ways: no more than eight AN/USQ-69 devices per channel is allowed; and no more than one input file and one output file can be attached to a single terminal.

Ada/L disk device drivers limit their support to peripheral equipment in the following ways: no more than one UYH-3 device per channel is allowed; and no more than four disks units per device. However, more than one open file is permitted on a disk.

```
*****************************************************************
PACKAGE TEXT_IO

-- C 1987 United States Government as represented by
--   the Secretary of the Navy. ALL RIGHTS RESERVED.

--   (The U.S. Government possesses the unlimited rights
--   throughout the world for government purposes to
--   publish, translate, reproduce, deliver, perform, and
--   dispose of the technical data, computer software, or
--   computer firmware contained herein; and to authorize
--   others to do so.)

--
--
-- Revision History:

--
--     3 Feb 1987  JGR
--         Package Specification Created.


WITH IO_EXCEPTIONS, ADA_RTLIB, IO_DEFS, FILE_IO;

PACKAGE TEXT_IO IS

PRAGMA PAGE;   -- In Package TEXT_IO Specification

--| JUSTIFICATION:
--|
--|     TEXT_IO provides input and output services for textual
--| files including creation, deletion, opening, and closing (as
--| described in Section 14.3.10 of [ANSI/MIL-STD-1815A]).
--|
--|     TEXT_IO also will make use of the Ada feature to overload
--| subprogram names.  In the cases where overloading is used,
--| each subprogram will be listed separately in its entirety.


--| DATA:
--|
--|     Data associated with this package will include the type
--| declarations for COUNT, POSITIVE_COUNT, FIELD, NUMBER_BASE,
--| TYPE_SET, FILE_MODE and the limited private type FILE_TYPE.
--| The FILE_TYPE is an access to the FILE_CONTROL_BLOCK, also
--| declared in this package specification.  Object for the
--| standard and current default input/output FILE_CONTROL_BLOCKs
--| are declared in this package specification.


TYPE file_type IS LIMITED PRIVATE;
   -- Forward reference of the private type.

TYPE file_mode IS
   -- The enumerations used to indicate whether a file is set
   -- for input or output.
   (in_file,    -- File is set for read only (input).
    out_file    -- File is set for write only (output).
   );

TYPE count IS RANGE 0..INTEGER'LAST;
   -- Implementation-dependent.
   -- This is the maximum allowable range on columns, lines,
   -- and pages.  Zero is used here to indicate special case of
   -- an empty item.

SUBTYPE positive_count IS text_io.count
   RANGE 1..text_io.count'LAST;
   -- Used to establish the allowable range for columns, lines,
   -- pages, and the current indices of each.

unbounded : CONSTANT text_io.count := 0;
            -- The line and page lengths used for initialization
            -- in the private implementation-dependent
            -- declarations.

SUBTYPE field IS INTEGER RANGE 0..INTEGER'LAST;
   -- Implementation-dependent.
   -- This is the allowable range for widths in the type fields.

SUBTYPE number_base IS INTEGER RANGE 2..16;
   -- Allowable range of numeric bases used in based literal
   -- integers.

TYPE type_set IS
   -- Determines which character set is used for identifiers.
   (lower_case,    -- Lower case characters.
    upper_case     -- Upper case characters.
   );

--| INITIALIZATION:
--|
--|     The elaboration of this package will initialize the
--| standard input and output devices of the Ada/L environment
--| and set the current input and output devices to the standard
--| input and output devices.


--| CREATED TASKS:
--|
--|     None.


--| NESTED PACKAGES:
--|
--|     TEXT_IO.INTEGER_IO,
--|     TEXT_IO.FLOAT_IO,
--|     TEXT_IO.FIXED_IO,
--|     TEXT_IO.ENUMERATION_IO
--| SUBPROGRAMS AND TASKS:
--|
--|


    PROCEDURE CREATE
    -- will create a file for text input-output.
       (file : IN OUT text_io.file_type;
               -- The pointer to the File_Control_Block.

        mode : IN text_io.file_mode := text_io.out_file;
               -- Specifies the direction of data transfer.

        name : IN STRING := "";
               -- Holds the external name of the file.

        form : IN STRING := ""
               -- System-dependent file characteristics.

       );


    PROCEDURE OPEN
    -- will a file for text input-output.
       (file : IN OUT text_io.file_type;
               -- The pointer to the File_Control_Block.

        mode : IN text_io.file_mode;
               -- Specifies the direction of data transfer.

        name : IN STRING;
               -- Holds the external name of the file.

        form : IN STRING := ""
               -- System-dependent file characteristics.

       );


    PROCEDURE CLOSE
    -- will close the text input-output file.
       (file : IN OUT text_io.file_type
               -- The pointer to the File_Control_Block.

       );


    PROCEDURE DELETE
    -- will delete the text input-output file.
       (file : IN OUT text_io.file_type
               -- The pointer to the File_Control_Block.

       );
```

```
PROCEDURE RESET
-- will reset a text input-output file and change its mode
-- to the requested mode.
   (file : IN OUT text_io.file_type;
           -- The pointer to the File_Control_Block.

    mode : IN text_io.file_mode
           -- The new mode of the file once it is reset.

   );


PROCEDURE RESET
-- will reset a text input-output file but will not change
-- its mode.
   (file : IN OUT text_io.file_type
           -- The pointer to the File_Control_Block.

   );


FUNCTION MODE
-- will return the mode of the given text input-output file.
   (file  : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN text_io.file_mode;
            -- Designator for the mode of the file.


FUNCTION NAME
-- will return the name of the given text input-output file.
   (file  : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN STRING;
            -- The external name of the file.


FUNCTION FORM
-- will return the form of the given text input-output file.
   (file  : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN STRING;
            -- The form of the file.


FUNCTION IS_OPEN
-- will return the status of the given input-output file.
   (file  : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN BOOLEAN;
```

```
            -- The pointer to the File_Control_Block.

     to    : IN text_io.count
            -- The requested new line length maximum.

    );


PROCEDURE SET_LINE_LENGTH
-- will set the maximum line length of the default text
-- output file to the given length.
    (to : IN text_io.count
          -- The requested new line length maximum.

    );


PROCEDURE SET_PAGE_LENGTH
-- will set the maximum page length of the given text output
-- file to the given length.
    (file : IN text_io.file_type;
            -- The pointer to the File_Control_Block.

     to    : IN text_io.count
            -- The requested new page length limit.

    );


PROCEDURE SET_PAGE_LENGTH
-- will set the maximum page length of the default text
-- output file to the given length.
    (to : IN text_io.count
          -- The requested new page length limit.

    );


FUNCTION LINE_LENGTH
-- will return the maximum line length for the given text
-- output file.
   (file  : IN text_io.file_type
             -- The pointer to the File_Control_Block.

    ) RETURN text_io.count;
             -- The maximum line length.


FUNCTION LINE_LENGTH
-- will return the maximum line length for the default text
-- output file.

    RETURN text_io.count;
           -- The maximum line length.
```

```
FUNCTION PAGE_LENGTH
-- will return the maximum page length for the given text
-- output file.
   (file  : IN text_io.file_type
          -- The pointer to the File_Control_Block.

   ) RETURN text_io.count;
          -- The maximum page count.


FUNCTION PAGE_LENGTH
-- will return the maximum page length for the default text
-- output file.

   RETURN text_io.count;
        -- The maximum line length.

PROCEDURE NEW_LINE
-- will put the requested amount of new lines in the given
-- text output file.
   (file    : IN text_io.file_type;
              -- The pointer to the File_Control_Block.

    spacing : IN text_io.positive_count := 1
              -- Number of lines to advance.  Initialized to
              -- one for a default value.  Hence, this
              -- parameter is optional.

   );


PROCEDURE NEW_LINE
-- will put the requested amount of new lines in the default
-- text output file.
   (spacing : IN text_io.positive_count := 1
              -- Number of lines to advance.  Initialized to
              -- one for a default value.  Hence, this
              -- parameter is optional.

   );


PROCEDURE SKIP_LINE
-- will skip the requested amount of lines in the given text
-- input file.
   (file    : IN text_io.file_type;
              -- The pointer to the File_Control_Block.

    spacing : IN text_io.positive_count := 1
              -- The number of lines to be advanced in the
              -- file.  The default of one is set to ensure
              -- that at least an advance to the beginning
              -- of the next line is achieved.
   );


PROCEDURE SKIP_LINE
-- will skip the requested amount of lines in the default
-- text input file.
   (spacing : IN text_io.positive_count := 1
              -- The number of lines to be advanced in the
              -- file.  The default of one is set to ensure
              -- that at least an advance to the beginning
              -- of the next line is achieved.

   );


FUNCTION END_OF_LINE
-- will indicate if the end of the line has been reached for
-- the given text input file.
   (file  : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN BOOLEAN;
            -- Indication of end of line found.


FUNCTION END_OF_LINE
-- will indicate if the end of the line has been reached for
-- the default text input file.

   RETURN BOOLEAN;
          -- Indication of end of line found.


PROCEDURE NEW_PAGE
-- will end the current page and start a new page in the
-- given text output file.
   (file : IN text_io.file_type
           -- The pointer to the File_Control_Block.

   );


PROCEDURE NEW_PAGE;
-- will end the current page and start a new page in the
-- default text output file.


PROCEDURE SKIP_PAGE
-- will skip to the beginning of the next page in the given
-- text input file.
   (file : IN text_io.file_type
           -- The pointer to the File_Control_Block.

   );
```

```
PROCEDURE SKIP_PAGE;
-- will skip to the beginning of the next page in the
-- default text input file.


FUNCTION END_OF_PAGE
-- will indicate if the end of the page has been reached
-- for the given text input file.
   (file   : IN text_io.file_type
             -- The pointer to the File_Control_Block.

   ) RETURN BOOLEAN;
             -- Indication of end of line found.


FUNCTION END_OF_PAGE
-- will indicate if the end of the page has been reached for
-- the default text input file.

   RETURN BOOLEAN;
           -- Indication of end of line found.


FUNCTION END_OF_FILE
-- will indicate if the end of the file has been reached for
-- the given text input file.
   (file   : IN text_io.file_type
             -- The pointer to the File_Control_Block.

   ) RETURN BOOLEAN;
             -- Indication of end of file found.


FUNCTION END_OF_FILE
-- will indicate if the end of the file has been reached for
-- the default text input file.

   RETURN BOOLEAN;
           -- Indication of end of file found.


PROCEDURE SET_COL
-- will set the current column to read-write to the given
-- column in the given text input-output file.
   (file : IN text_io.file_type;
           -- The pointer to the File_Control_Block.

    to   : IN text_io.positive_count
           -- The new column to set the position pointer to.

   );


PROCEDURE SET_COL
```

```
-- will set the current column to read-write to the given
-- column in the default text input-output file.
   (to : IN text_io.positive_count
         -- The new column to set the position pointer to.

   );


PROCEDURE SET_LINE
-- will set the current line to read-write to the given line
-- in the given text input-output file.
   (file : IN text_io.file_type;
           -- The pointer to the File_Control_Block.

    to   : IN text_io.positive_count
           -- The new line to set the position pointer to.

   );


PROCEDURE SET_LINE
-- will set the current line to read-write to the given line
-- in the default text input-output file.
   (to : IN text_io.positive_count
         -- The new line to set the position pointer to.

   );


FUNCTION COL
-- will return the current column position for the given text
-- input-output file.
   (file  : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN text_io.positive_count;
            -- The value of the current column index.


FUNCTION COL
-- will return the current column position for the default
-- text input-output file.

   RETURN text_io.positive_count;
          -- The value of the current column index.


FUNCTION LINE
-- will return the current line position for the given text
-- input-output file.
   (file  : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN text_io.positive_count;
            -- The value of the current line index.
```

```
FUNCTION LINE
-- will return the current line position for the default text
-- input-output file.

   RETURN text_io.positive_count;
          -- The value of the current line index.


FUNCTION PAGE
-- will return the current page number for the given text
-- input-output file.
   (file   : IN text_io.file_type
            -- The pointer to the File_Control_Block.

   ) RETURN text_io.positive_count;
            -- The current page number.


FUNCTION PAGE
-- will return the current page number for the default text
-- input-output file.

   RETURN text_io.positive_count;
          -- The current page number.


PROCEDURE GET
-- will read a character from the given text input file.
   (file : IN text_io.file_type;
           -- The pointer to the File_Control_Block.

    item : OUT CHARACTER
           -- The character to return.

   );


PROCEDURE GET
-- will read a character from the default text input file.
   (item : OUT CHARACTER
           -- The character to return.

   );


PROCEDURE PUT
-- will write a character to the given text output file.
   (file : IN text_io.file_type;
           -- The pointer to the File_Control_Block.

    item : IN CHARACTER
           -- Character to write to the file.
   );


PROCEDURE PUT
-- will write a character to the default text output file.
   (item : IN CHARACTER
           -- Character to write to the File_Control_Block.

   );


PROCEDURE GET
-- will read a string from the given text input file.
   (file : IN text_io.file_type;
           -- The pointer to the File_Control_Block.

    item : OUT STRING
           -- The string to return.

   );


PROCEDURE GET
-- will read a string from the default text input file.
   (item : OUT STRING
           -- The string to return.

   );


PROCEDURE PUT
-- will write a string to the given text output file.
   (file : IN text_io.file_type;
           -- The pointer to the File_Control_Block.

    item : IN STRING
           -- String to write to the file.

   );


PROCEDURE PUT
-- will write a string to the default text output file.
   (item : IN STRING
           -- String to write to the File_Control_Block.

   );


PROCEDURE GET_LINE
-- will read the remaining portion of a line from the given
-- text input file.
   (file : IN text_io.file_type;
           -- The pointer to the File_Control_Block.
```

```
         item : OUT STRING;
                -- The string to return.

         last : OUT NATURAL
                -- An index containing a value such that
                -- item(last) is last character read.

        );


    PROCEDURE GET_LINE
    -- will read the remaining portion of a line from the
    -- default text input file.
        (item : OUT STRING;
                -- The string to return.

         last : OUT NATURAL
                -- An index containing a value such that
                -- item(last) is last character read.

        );


    PROCEDURE PUT_LINE
    -- will write a line to the given text output file and
    -- advance to the next line.
        (file : IN text_io.file_type;
                -- The pointer to the File_Control_Block.

         item : IN STRING
                -- String to write to the file.

        );


    PROCEDURE PUT_LINE
    -- will write a line to the default text output file and
    -- advance to the next line.
        (item : IN STRING
                -- String to write to the File_Control_Block.

        );

PRAGMA PAGE;   -- In Package TEXT_IO.INTEGER_IO Specification

-- C 1987 United States Government as represented by
--   the Secretary of the Navy. ALL RIGHTS RESERVED.

--   (The U.S. Government possesses the unlimited rights
--   throughout the world for government purposes to
--   publish, translate, reproduce, deliver, perform, and
--   dispose of the technical data, computer software, or
--   computer firmware contained herein; and to authorize
--   others to do so.)
```

```
--
--
-- Revision History:
--
--      3 Feb 1987  JGR
--          Package Specification Created.

GENERIC

   TYPE num IS RANGE <>;
      -- The type and range used upon instantiation of the
      -- INTEGER_IO package.


PACKAGE INTEGER_IO IS

PRAGMA PAGE;    -- In Package TEXT_IO.INTEGER_IO Specification

--| JUSTIFICATION:
--|
--|     INTEGER_IO contains the subprograms necessary for the
--| user to perform Text I/O for integer types (as described in
--| Section 14.3.7 of [ANSI/MIL-STD-1815A]).  INTEGER_IO is a
--| generic package, internal to the body of TEXT_IO and must be
--| instantiated prior to its use. INTEGER_IO primarily allows
--| the reading (getting) and writing (putting) of integers of
--| the type INTEGER_IO.NUM either with respect to strings or
--| with respect to text files.
--|
--|     INTEGER_IO also will make use of the Ada feature to
--| overload subprogram names.  In the cases where overloading
--| is used, each subprogram will be listed separately in its
--| entirety.


--| DATA:
--|
--|     Generic package level declarations for input-output of
--| integer types.


default_width : text_io.field := num'WIDTH;
                -- The default integer width.  Initialized to
                -- the width of the instantiated type, though
                -- the user may reset to the desired width.

default_base  : text_io.number_base := 10;
                -- The default base width.  Initialized to the
                -- width of the instantiated type, though the
                -- user may reset to the desired width.


--| CREATED TASKS:
```

```
--|
--|     None.


--| INITIALIZATION:
--|
--|     The data of this package specification is initialized to
--| a default integer type NUM with a default width large enough
--| to support the range of NUM and a default base of ten.
--|
--|     Elaboration of this package will raise an USE_ERROR if
--| the size of NUM exceeds the implementation dependent size
--| of a long_integer.


--| SUBPROGRAMS AND TASKS:
--|
--|


   PROCEDURE GET
   -- will read an integer from the given text input file.
      (file  : IN text_io.file_type;
               -- Pointer to the specified file to read from.

       item  : OUT num;
               -- The generic integer type result.

       width : IN text_io.field := 0
               -- Amount of characters to read.  The default is
               -- zero and will read the entire string.

      );


   PROCEDURE GET
   -- will read an integer from the default text input file.
      (item  : OUT num;
               -- The generic integer type result.

       width : IN text_io.field := 0
               -- Amount of characters to read.  The default is
               -- zero and will read the entire string.

      );


   PROCEDURE PUT
   -- will write an integer to the given text input file.
      (file  : IN text_io.file_type;
               -- Pointer to the specified file to write to.

       item  : IN num;
               -- The generic integer type to write.
```

```
    width : IN text_io.field := default_width;
            -- The width of item.  Initialized to the width
            -- of the instantiated integer type.  May be
            -- reset by the user.

    base  : IN text_io.number_base := default_base
            -- The base of item.  Initialized to the default
            -- base 10, but may have ranges 2..16.

   );


PROCEDURE PUT
-- will write an integer to the default text input file.
   (item  : IN num;
            -- The generic integer type to write.

    width : IN text_io.field := default_width;
            -- The width of item.  Initialized to the width
            -- of the instantiated integer type.  May be
            -- reset by the user.

    base  : IN text_io.number_base := default_base
            -- The base of item.  Initialized to the default
            -- base 10, but may have ranges 2..16.

   );


PROCEDURE GET
-- will read an integer form the given text string.
   (from : IN STRING;
           -- The string to read from.

    item : OUT num;
           -- The generic integer type result.

    last : OUT positive
           -- Index of the last character read
           -- from the string.

   );


PROCEDURE PUT
-- will write an integer to the given text string.
   (to   : OUT STRING;
           -- The string containing the integer image.

    item : IN num;
           -- The generic integer type to write.

    base : IN text_io.number_base := default_base
           -- The base of item.  Initialized to the default
           -- base 10, but may have ranges 2..16.
```

```
      );

END INTEGER_IO;

PRAGMA PAGE;   -- In Package TEXT_IO.FLOAT_IO Specification

-- C 1987 United States Government as represented by
--   the Secretary of the Navy. ALL RIGHTS RESERVED.

--   (The U.S. Government possesses the unlimited rights
--   throughout the world for government purposes to
--   publish, translate, reproduce, deliver, perform, and
--   dispose of the technical data, computer software, or
--   computer firmware contained herein; and to authorize
--   others to do so.)

--
--
-- Revision History:

--
--     3 Feb 1987  JGR
--         Package Specification Created.


GENERIC

   TYPE num IS DIGITS <>;
      -- The type and range used upon instantiation of the
      -- FLOAT_IO package.


PACKAGE FLOAT_IO IS

PRAGMA PAGE;   -- In Package TEXT_IO.FLOAT_IO Specification

--| JUSTIFICATION:
--|
--|     FLOAT_IO contains the subprograms necessary for the user
--| to perform Text_IO for floating point types (as described in
--| Section 14.3.8 of [ANSI/MIL-STD-1815A]).  FLOAT_IO is a
--| generic package, internal to the body of TEXT_IO and must be
--| instantiated prior to its use.  FLOAT_IO primarily allows
--| the reading (getting) and writing (putting) of floating
--| point values of the type FLOAT_IO.NUM either with respect to
--| strings or with respect to text files.
--|
--|     FLOAT_IO also will make use of the Ada feature to overload
--| subprogram names.  In the cases where overloading is used,
--| each subprogram will be listed separately in its entirety.


--| DATA:
--|
--|     Generic package level declarations for input-output of
--| floating point types.


default_fore : text_io.field := 2;
               -- The default width of the whole number portion
               -- of the floating point type.

default_aft  : text_io.field := num'DIGITS-1;
               -- The default width of the decimal portion of the
               -- floating point type.

default_exp  : text_io.field := 3;
               -- The default width of the exponent field
               -- following the character E when a nonzero
               -- exponent is provided.

--| CREATED TASKS:
--|
--|     None.


--| INITIALIZATION:
--|
--|     The data of this package specification is initialized to
--| a default floating point type NUM with a default FORE of two
--| characters (the decimal representation), a default AFT
--| large enough to support the range of NUM (the fractional
--| representation), and a default EXP of three characters (the
--| exponent representation).


--| SUBPROGRAMS AND TASKS:
--|
--|


   PROCEDURE GET
   -- will read a floating point real from the given text
   -- input file.
      (file  : IN text_io.file_type;
               -- Pointer to the specified file to read from.

       item  : OUT num;
               -- The generic floating point type result.

       width : IN text_io.field := 0
               -- Amount of characters to read.  The default is
               -- zero and will read the entire string.

      );


   PROCEDURE GET
   -- will read a floating point real from the default text
-- input file.
   (item  : OUT num;
            -- The generic floating point type result.

    width : IN text_io.field := 0
            -- Amount of characters to read.  The default is
            -- zero and will read the entire string.

   );


PROCEDURE PUT
-- will write a floating point real to the given text input
-- file.
   (file : IN text_io.file_type;
           -- Pointer to the specified file to write to.

    item : IN num;
           -- The generic floating point type to write.

    fore : IN text_io.field := default_fore;
           -- The width of the whole number portion of the
           -- floating point value.  Initialized to the width
           -- of two.  May be reset by the user.

    aft  : IN text_io.field := default_aft;
           -- The width of the decimal portion of the
           -- floating point value.  Initialized to the
           -- default width of the number of digits in the
           -- instantiated type minus one.  May be reset by
           -- the user.

    exp  : IN text_io.field := default_exp
           -- The width of the exponent field following the
           -- character E.  Initialized to the width of
           -- three.  May be reset by the user.

   );


PROCEDURE PUT
-- will write a floating point real to the default text input
-- file.
   (item : IN num;
           -- The generic floating point type to write.

    fore : IN text_io.field := default_fore;
           -- The width of the whole number portion of the
           -- floating point value.  Initialized to the width
           -- of two.  May be reset by the user.

    aft  : IN text_io.field := default_aft;
           -- The width of the decimal portion of the
           -- floating point value.  Initialized to the
           -- default width of the number of digits in the
```

```
                -- instantiated type minus one.  May be reset by
                -- the user.

         exp  : IN text_io.field := default_exp
                -- The width of the exponent field following the
                -- character E.  Initialized to the width of
                -- three.  May be reset by the user.

        );


    PROCEDURE GET
    -- will read a floating point real from the given text string.
       (from : IN STRING;
                -- The string to read from.

         item : OUT num;
                -- The generic floating point type result.

         last : OUT positive
                -- Index of the last character read from
                -- the string.

        );


    PROCEDURE PUT
    -- will write a floating point real to the given text string.
       (to   : OUT STRING;
                -- The string containing the floating point image.

         item : IN num;
                -- The generic floating point type to write.

         aft  : IN text_io.field := default_aft;
                -- The width of the decimal portion of the
                -- floating point value.  Initialized to the
                -- default width of the number of digits in the
                -- instantiated type minus one.  May be reset by
                -- the user.

         exp  : IN text_io.field := default_exp
                -- The width of the exponent field following the
                -- character E.  Initialized to the width of
                -- three.  May be reset by the user.

        );

END FLOAT_IO;

PRAGMA PAGE;    -- In Package TEXT_IO.FIXED_IO Specification

-- C 1987 United States Government as represented by
--    the Secretary of the Navy. ALL RIGHTS RESERVED.
--   (The U.S. Government possesses the unlimited rights
--   throughout the world for government purposes to
--   publish, translate, reproduce, deliver, perform, and
--   dispose of the technical data, computer software, or
--   computer firmware contained herein; and to authorize
--   others to do so.)

--
--
-- Revision History:

--
--      3 Feb 1987  JGR
--          Package Specification Created.


GENERIC

   TYPE num IS DELTA <>;
      -- The type and range used upon instantiation of the
      -- FIXED_IO package.


PACKAGE FIXED_IO IS

PRAGMA PAGE;    -- In Package TEXT_IO.FIXED_IO Specification

--| 1. JUSTIFICATION
--|
--|     FIXED_IO contains the subprograms necessary for the user
--| to perform Text_IO for fixed point types (as described in
--| Section 14.3.8 of [ANSI/MIL-STD-1815A]).  FIXED_IO is a
--| generic package, internal to the body of TEXT_IO and must be
--| instantiated prior to its use. FIXED_IO primarily allows the
--| reading (getting) and writing (putting) of fixed point values
--| of the type FIXED_IO.NUM either with respect to strings or
--| with respect to text files.
--|
--|     FIXED_IO also will make use of the Ada feature to
--| overload subprogram names.  In the cases where overloading
--| is used, each subprogram will be listed separately in its
--| entirety.


--| 2. DATA
--|
--|     Generic package level declarations for input-output of
--| fixed point types.


default_fore : text_io.field := num'FORE;
               -- The default width of the whole number portion
               -- of the floating point type.

default_aft  : text_io.field := num'AFT;
```

```
                  -- The default width of the decimal portion of the
                  -- floating point type.

default_exp     : text_io.field := 0;
                  -- The default width of the exponent field
                  -- following the character E when a nonzero
                  -- exponent is provided.


--| 3. CREATED TASKS
--|
--|     None.


--| 4. INITIALIZATION
--|
--|     The data of this package specification is ini'  lized to
--| a default fixed point type NUM with a default FORE large
--| enough to represent the decimal part of type NUM, a default
--| AFT large enough to represent the fractional part of type
--| NUM, and a default EXPonent of zero characters.


--| 5. SUBPROGRAMS AND TASKS
--|


-- The following routines are used for fixed point input-output.

   PROCEDURE GET
   -- will read a fixed point real from the given text input
   -- file.
      (file  : IN text_io.file_type;
               -- Pointer to the specified file to read from.

       item  : OUT num;
               -- The generic fixed point type result.

       width : IN text_io.field := 0
               -- Amount of characters to read.  The default is
               -- zero and will read the entire string.

      );


   PROCEDURE GET
   -- will read a fixed  point real from the default text
   -- input file.
      (item  : OUT num;
               -- The generic fixed point type result.

       width : IN text_io.field := 0
               -- Amount of characters to read.  The default is
               -- zero and will read the entire string.
   );


  PROCEDURE PUT
  -- will write a fixed point real to the given text input file.
     (file : IN text_io.file_type;
             -- Pointer to the specified file to write to.

      item : IN num;
             -- The generic fixed point type to write.

      fore : IN text_io.field := default_fore;
             -- The width of the whole number portion of the
             -- fixed point value.  Initialized to the width
             -- of two.  May be reset by the user.

      aft  : IN text_io.field := default_aft;
             -- The width of the decimal portion of the fixed
             -- point value.  Initialized to the default width
             -- of the number of digits in the instantiated
             -- type minus one.  May be reset by the user.

      exp  : IN text_io.field := default_exp
             -- The width of the exponent field following the
             -- character E.  Initialized to the width of
             -- three.  May be reset by the user.

   );


  PROCEDURE PUT
  -- will write a fixed point real to the default text input
  -- file.
     (item : IN num;
             -- The generic fixed point type to write.

      fore : IN text_io.field := default_fore;
             -- The width of the whole number portion of the
             -- fixed point value.  Initialized to the width
             -- of two.  May be reset by the user.

      aft  : IN text_io.field := default_aft;
             -- The width of the decimal portion of the fixed
             -- point value.  Initialized to the default width
             -- of the number of digits in the instantiated
             -- type minus one.  May be reset by the user.

      exp  : IN text_io.field := default_exp
             -- The width of the exponent field following the
             -- character E.  Initialized to the width of
             -- three.  May be reset by the user.

   );
```

```
    PROCEDURE GET
    -- will read a fixed point real from the given text string.
       (from : IN STRING;
               -- The string to read from.

        item : OUT num;
               -- The generic fixed point type result.

        last : OUT positive
               -- Index of the last character read from
               -- the string.

       );


    PROCEDURE PUT
    -- will write a fixed point real to the given text string.
       (to   : OUT STRING;
               -- The string containing the fixed point image.

        item : IN num;
               -- The generic fixed point type to write.

        aft  : IN text_io.field := default_aft;
               -- The width of the decimal portion of the fixed
               -- point value.  Initialized to the default width
               -- of the number of digits in the instantiated
               -- type minus one.  May be reset by the user.

        exp  : IN text_io.field := default_exp
               -- The width of the exponent field following the
               -- character E.  Initialized to the width of
               -- three.  May be reset by the user.

       );

END FIXED_IO;

PRAGMA PAGE;   -- In Package TEXT_IO.ENUMERATION_IO Specification

-- C 1987 United States Government as represented by
--    the Secretary of the Navy. ALL RIGHTS RESERVED.

--    (The U.S. Government possesses the unlimited rights
--    throughout the world for government purposes to
--    publish, translate, reproduce, deliver, perform, and
--    dispose of the technical data, computer software, or
--    computer firmware contained herein; and to authorize
--    others to do so.)

--
--
-- Revision History:

--
```

```
--     3 Feb 1987  JGR
--         Package Specification Created.


GENERIC

   TYPE enum IS (<>);
      -- The type used upon instantiation of the
      -- ENUMERATION_IO package.


PACKAGE ENUMERATION_IO IS

PRAGMA PAGE;    -- In Package TEXT_IO.ENUMERATION_IO Specification

--| JUSTIFICATION:
--|
--|     ENUMERATION_IO contains the subprograms necessary for the
--| user to perform Text_IO for enumeration types (as described
--| in Section 14.3.9 of [ANSI/MIL-STD-1815A]).  ENUMERATION_IO
--| is a generic package, internal to the body of TEXT_IO and
--| must be instantiated prior to its use.  ENUMERATION_IO
--| primarily allows the reading (getting) and writing (putting)
--| of enumerations of the type ENUMERATION_IO.ENUM either with
--| respect to strings or with respect to text files.
--|
--|     ENUMERATION_IO also will make use of the Ada feature to
--| overload subprogram names.  In the cases where overloading is
--| used, each subprogram will be listed separately in its
--| entirety.


--| DATA:
--|
--|     Generic package level declarations for input-output of
--| enumeration types.


default_width   : text_io.field := 0;
                  -- The default field width of the character(s)
                  -- including any trailing spaces.

default_setting : text_io.type_set := text_io.upper_case;
                  -- The default character case of the letter(S).


--| CREATED TASKS:
--|
--|     None.


--| INITIALIZATION:
--|
--|     The data in this package specification is initialized to
--| a default enumeration type NUM with a default width of zero
--| and a default case of upper_case.


--| SUBPROGRAMS AND TASKS:
--|


    PROCEDURE GET
    -- will read an enumeration literal from the given text
    -- input file.
       (file : IN text_io.file_type;
               -- Pointer to the specified file to read from.

        item : OUT enum
               -- The generic enumeration type result.

       );


    PROCEDURE GET
    -- will read an enumeration literal from the default text
    -- input file.
       (item : OUT enum
               -- The generic enumeration type result.

       );


    PROCEDURE PUT
    -- will write an enumeration literal to the given text input
    -- file.
       (file  : IN text_io.file_type;
                -- Pointer to the specified file to write to.

        item  : IN enum;
                -- The generic enumeration type to write.

        width : IN text_io.field := default_width;
                -- The width of item.  Initialized to the width
                -- of the instantiated enumeration type.  May be
                -- reset by the user.

        set   : IN text_io.type_set := default_setting
                -- The character type to be used.  Initialized to
                -- the default setting of upper-case letters.
                -- May be reset by the user.

       );


    PROCEDURE PUT
    -- will write an enumeration literal to the default text
    -- input file.
       (item  : IN enum;
                -- The generic enumeration type to write.
```

```
          width : IN text_io.field := default_width;
                  -- The width of item.  Initialized to the width
                  -- of the instantiated enumeration type.  May be
                  -- reset by the user.

          set   : IN text_io.type_set := default_setting
                  -- The character type to be used.  Initialized to
                  -- the default setting of upper-case letters.
                  -- May be reset by the user.

        );


    PROCEDURE GET
    -- will read an enumeration literal from the given text
    -- string.
       (from : IN STRING;
               -- The string to read from.

        item : OUT enum;
               -- The generic enumeration type result.

        last : OUT positive
               -- Index of the last character of the enumeration
               -- literal.

       );


    PROCEDURE PUT
    -- will write an enumeration literal to the given text string.
       (to   : OUT STRING;
               -- The string containing the enumeration image.

        item : IN enum;
               -- The generic enumeration type to write.

        set  : IN text_io.type_set := default_setting
               -- The character type to be used.  Initialized to
               -- the default setting of upper-case letters.  May
               -- be reset by the user.

    );

END ENUMERATION_IO;

PRAGMA PAGE;    -- In Package TEXT_IO Specification

-- These are the Text I/O-specific Ada exceptions.

status_error : EXCEPTION RENAMES io_exceptions.status_error;
               -- Indicates that the file is not properly set
               -- for the requested operation.
```

```
mode_error    : EXCEPTION RENAMES io_exceptions.mode_error;
                -- Indicates that the file was not in the proper
                -- mode for an attempted read or write operation.

name_error    : EXCEPTION RENAMES io_exceptions.name_error;
                -- Indicates that the syntax of the name is not
                -- valid or that the name has already been used.

use_error     : EXCEPTION RENAMES io_exceptions.use_error;
                -- Indicates the improper usage of a file.

device_error : EXCEPTION RENAMES io_exceptions.device_error;
                -- Indicates that a device driver program has
                -- failed for one reason or another.

end_error     : EXCEPTION RENAMES io_exceptions.end_error;
                -- Indicates that an attempt was made to read
                -- past the end of the file.

data_error    : EXCEPTION RENAMES io_exceptions.data_error;
                -- Indicates that an attempt was made, to read
                -- from a file into a buffer or write from a
                -- buffer into a file, data that is of the wrong
                -- type.

layout_error : EXCEPTION RENAMES io_exceptions.layout_error;
                -- Indicates that an attempt was made to set the
                -- column or line numbers in excess of the
                -- currently specified line and page maximums.
                -- It will also indicate when an attempt has
                -- been made to PUT too many characters to a
                -- string.

PRAGMA PAGE;    -- In Package TEXT_IO Specification

-- The following are the private implementation-dependent
-- declarations.

PRIVATE

TYPE file_control_block;
   -- The File_Control_Block forward declaration.

TYPE file_type IS ACCESS text_io.file_control_block;
   -- Defines the access pointer to the File_Control_Block.

-- These are the required file marker values for this particular
-- implementation.

line_term           : CONSTANT CHARACTER := ASCII.LF;
                      -- The character used to indicate the end
                      -- of the current line.  <CTRL-M>.

page_term           : CONSTANT CHARACTER := ASCII.FF;
                      -- The character used to indicate the end
```

```
                        -- of the current page.  <CTRL-L>.

file_term             : CONSTANT CHARACTER := ASCII.SUB;
                        -- The character used to indicate the end
                        -- of the file. <CTRL-Z>.


-- The following constant declarations are required for use
-- within the File_Control_Block.

null_strm             : CONSTANT file_io.stream_id_prv := NULL;
                        -- A null value for the stream pointers
                        -- used during initializations.

buffer_length         : CONSTANT := io_defs.buffer_data_limit;
                        -- The upper bound for the text buffers
                        -- used in the file_Control_Block.  This
                        -- is the maximum amount of data that the
                        -- vertual mamory can handle in an I/O
                        -- request at one time.

max_line_length       : CONSTANT := io_defs.buffer_data_limit;
                        -- Establishes the restrictions for a line
                        -- length of 1020.  This will then
                        -- coincide with the maximum buffer_length
                        -- and the maximum amount of data that
                        -- the vertual memory can handle in an
                        -- I/O request at one time and still
                        -- leave the necessary amount of trailing
                        -- space for the information required by
                        -- the Ada/L device drivers.


-- The folowing type declarations define types used within the
-- File_Control_Block.

TYPE char_buffer IS ARRAY (io_defs.data_length_int RANGE
   1..text_io.buffer_length) OF CHARACTER;
   -- Estsblishes the characteristics for the text
   -- buffers used in the File_Control_Block.

TYPE buffer_ptr IS ACCESS text_io.char_buffer;
   -- Defines the access pointers to the text buffers
   -- used in the File_Control_Block.


-- The following object declaration is required to support
-- temporary file name creation.

temp_file_count : INTEGER := 0;
                  -- Counts the number of temporary files created
                  -- by the User-Written_Ada_Program.  The 'IMAGE
                  -- of this object will be added to the
                  -- temporaryfile name to ensure that all files
                  -- remain unique.
-- These are the standard and current file control blocks. They
-- are not visible to the user except through the provided
-- routines.

std_input       : text_io.file_type;
                  -- The access pointer to the Standard default
                  -- input File_Control_Block.

std_output      : text_io.file_type;
                  -- The access pointer to the Standard default
                  -- output File_Control_Block.

curr_input      : text_io.file_type;
                  -- The access pointer to the current default
                  -- input File_Control_Block.

curr_output     : text_io.file_type;
                  -- The access pointer to the current default
                  -- output File_Control_Block.


-- Full declaration for the File_Control_Block.
TYPE file_control_block IS
   -- The File_Control_Block state description.  The actual
   -- FILE_TYPE declarations will be access types to this record.
   RECORD
      strm_ptr        : file_io.stream_id_prv
                        := text_io.null_strm;
                        -- Contains the system's link to the
                        -- external file.

      external_name   : io_defs.file_name_str
                        := (OTHERS => ' ');
                        -- Holds the external name of the file.
                        -- Initialized to blanks to clear any
                        -- portion that will not be used.

      temporary       : BOOLEAN := FALSE;
                        -- Indicates whether the file was
                        -- created as a temporary file, or
                        -- created for permanent storage.
                        -- Initialized to the value of false
                        -- since the standard input and output
                        -- blocks also use this
                        -- File_Control_Block.

      files_class     : io_defs.file_class_enu
                        := io_defs.fc_text;
                        -- Contains the value indicating the
                        -- type of data in the file.  Initialized
                        -- for the text oriented input and
                        -- output.
```

```
f_mode          : text_io.file_mode;
                  -- Contains the value indicating the
                  -- mode of the file.

interactive     : BOOLEAN := FALSE;
                  -- Indicator showing whether the file is
                  -- being used either directly to or from
                  -- a display terminal, or to or from a
                  -- tape file.  Initialized to false fcr
                  -- tape files.

end_info        : BOOLEAN := FALSE;
                  -- Indicator showing whether the end of
                  -- data presently being processed has
                  -- been reached.

curr_col        : text_io.count := 1;
                  -- Holds the present column index
                  -- position for the current line.
                  -- Initialized to the start of a new
                  -- line.

curr_line       : text_io.count := 1;
                  -- Holds the present line index position
                  -- for the current page.  Initialized to
                  -- the start of a new page.

line_len        : text_io.count := text_io.unbounded;
                  -- Holds the maximum allowable line
                  -- length limit. Initialized for
                  -- unformatted text output.

curr_page       : text_io.count := 1;
                  -- Holds the present page index in the
                  -- current file. Initialized to the
                  -- start of a new file.

page_len        : text_io.count := text_io.unbounded;
                  -- Holds the maximum allowable page
                  -- length limit. Initialized for
                  -- unformatted text output.

eoln_found      : BOOLEAN := FALSE;
                  -- Indicator showing whether the end of
                  -- current line has been reached.
                  -- Initialized to false for the
                  -- beginning of the first line.

eop_found       : BOOLEAN := FALSE;
                  -- Indicator showing whether the end of
                  -- current page has been reached.
                  -- Initialized to false for the beginning
                  -- of the first page.

eof_found       : BOOLEAN := FALSE;
```

```
                                 -- Indicator showing whether the end of
                                 -- current file has been reached.
                                 -- Initialized to false for the
                                 -- beginning of a file.

         text_buf              : text_io.buffer_ptr := NULL;
                                 -- The primary text buffer for all
                                 -- reading and writing of text
                                 -- characters.

         next_buf              : text_io.buffer_ptr := NULL;
                                 -- The alternate text buffer for the
                                 -- reading of text characters.

         text_index            : io_defs.data_length_int := 0;
                                 -- Holds the value used to index into
                                 -- the primary text buffer.  Initialized
                                 -- to show that the primary text buffer
                                 -- is initially empty.

         curr_rec_length       : io_defs.data_length_int := 0;
                                 -- Holds the length of data, in elements,
                                 -- contained in the primary text buffer.
                                 -- Initialized to show that the primary
                                 -- text buffer is initially empty.

         next_rec_length       : io_defs.data_length_int := 0;
                                 -- Holds the length of data, in elements,
                                 -- contained in the alternate text
                                 -- buffer. Initialized to show that the
                                 -- alternate text buffer is initially
                                 -- empty.

         max_rec_length        : text_io.count := text_io.max_line_length;
                                 -- Holds the maximum allowable length of
                                 -- data, in elements, for the primary
                                 -- and alternate text buffers.
                                 -- Initialized to the maximum amount of
                                 -- elements that can be handled by the
                                 -- device drivers in a single transfer.

         exclusion             : ada_rtlib.mutex.semaphore_type;
                                 -- Declares the access to the mutual
                                 -- exclusion task required for solitary
                                 -- access to the File_Control_Block.
      END RECORD;

   END TEXT_IO;
```

```
**********************************************************************
PACKAGE DIRECT_IO

-- C 1987 United States Government as represented by
--   the Secretary of the Navy. ALL RIGHTS RESERVED.

--   (The U.S. Government possesses the unlimited rights
--   throughout the world for government purposes to
--   publish, translate, reproduce, deliver, perform, and
--   dispose of the technical data, computer software, or
--   computer firmware contained herein; and to authorize
--   others to do so.)

--
--
-- Revision History:

--
--     3 Feb 1987  JGR
--         Package Specification Created.


WITH IO_EXCEPTIONS, IO_SUPPORT;


GENERIC

    TYPE element_type IS PRIVATE;
       -- The type of the element instantiated on the call to
       -- any of the subprograms of this package.  It is
       -- determined by the type of the element being used
       -- in the invoking subprogram.


PACKAGE DIRECT_IO IS

PRAGMA PAGE;   -- In Package DIRECT_IO Specification

--| JUSTIFICATION:
--|
--|     DIRECT_IO contains the subprograms necessary for the user
--| to perform direct access I/O operations (as described in
--| Section 14.2.1 and Section 14.2.4 of [ANSI/MIL-STD-1815A]).
--| DIRECT_IO is primarily an interface, between the
--| User-Written_Ada_Program and the package IO_SUPPORT which
--| does the actual work for DIRECT_IO operations.  In serving
--| as this interface, DIRECT_IO contains only the calls and
--| necessary conversions for the calls to invoke the subprograms
--| of IO_SUPPORT.  DIRECT_IO is a generic package thus enabling
--| it to work for all instantiated types.
--|
--|     DIRECT_IO also will make use of the Ada feature to
--| overload subprogram names.  In the cases where overloading
--| is used, each subprogram will be listed separately in its
--| entirety.
--| DATA:
--|
--|     Type declarations for the limited private type FILE_TYPE
--| and the types FILE_MODE, COUNT, and POSITIVE_COUNT are
--| declared in this package specification.


TYPE file_type IS LIMITED PRIVATE;
   -- Forward declaration.  The formal declaration will
   -- appear later.

TYPE file_mode IS
   -- Allowable modes for direct files.
   (in_file,       -- Input mode (read only).
    inout_file,    -- Input and output modes (read
                   -- and write both).
    out_file       -- Output mode (write only).
   );

TYPE count IS RANGE 0..io_support.count'LAST;
   -- This is the maximum allowable range on columns, lines, and
   -- pages. Zero is used here to indicate special case of empty
   -- items.

SUBTYPE positive_count IS count
   RANGE 1..count'LAST;
   -- Used to establish the allowable range for buffer indices.


--| INITIALIZATION:
--|
--|     None.


--| CREATED TASKS:
--|
--|     None.


--| SUBPROGRAMS AND TASKS:
--|
--|


   PROCEDURE CREATE
   -- will create a file for direct access.
      (file : IN OUT file_type;
              -- Points to the block containing
              -- the file information.

       mode : IN file_mode := inout_file;
              -- Specifies that the file is to be both read from
              -- and written to.
```

```
     name : IN STRING := "";
            -- The name of the external file.

     form : IN STRING := ""
            -- Used by the system for file characteristics.

    );


 PROCEDURE OPEN
 -- will open a file for direct access.
    (file : IN OUT file_type;
            -- Points to the block containing the
            -- file information.

     mode : IN file_mode;
            -- Specifies whether the file is to be read from,
            -- written to, or both.

     name : IN STRING;
            -- The name of the external file.

     form : IN STRING := ""
            -- Used by the system for file characteristics.

    );


 PROCEDURE CLOSE
 -- will close a direct access file.
    (file : IN OUT file_type
            -- Points to the block containing the
            -- file information.

    );


 PROCEDURE DELETE
 -- will delete a direct access file.
    (file : IN OUT file_type
            -- Points to the block containing the
            -- file information.

    );


 PROCEDURE RESET
 -- will reset a direct access file and change
 -- its mode to the requested mode.
    (file : IN OUT file_type;
            -- Points to the block containing the
            -- file information.

     mode : IN file_mode
```

```
            -- Specifies whether the file is to be read from,
            -- written to, or both.

   );


PROCEDURE RESET
-- will reset a direct access file but will
-- not change its mode.
   (file : IN OUT file_type
            -- Points to the block containing the
            -- file information.

   );



FUNCTION MODE
-- gives the mode of the direct access file.
   (file  : IN file_type
             -- Points to the block containing the
             -- file information.

   ) RETURN file_mode;
             -- Mode to return to invoking subprogram.


FUNCTION NAME
-- gives the external name of the direct access file.
   (file  : IN file_type
             -- Points to the block containing the
             -- file information.

   ) RETURN STRING;
             -- The name of the external file.


FUNCTION FORM
-- gives the form of the direct access file.
   (file  : IN file_type
             -- Points to the block containing the
             -- file information.

   ) RETURN STRING;
             -- The form of the external file.


FUNCTION IS_OPEN
-- indicates whether the direct access file is open.
   (file  : IN file_type
             -- Points to the block containing the
             -- file information.

   ) RETURN BOOLEAN;
             -- Status of the file.
```

```
PROCEDURE READ
-- will read from a direct access file starting at the given
-- position.
   (file : IN file_type;
           -- Points to the block containing the
           -- file information.

    item : OUT element_type;
           -- Holds the data being read.

    from : IN positive_count
           -- Specifies the index to be used by the operation.

   );


PROCEDURE READ
-- will read from a direct access file from the position
-- following the last read position.
   (file : IN file_type;
           -- Points to the block containing the
           -- file information.

    item : OUT element_type
           -- Holds the data being read.

   );


PROCEDURE WRITE
-- will write to a direct access file at the position given.
   (file : IN file_type;
           -- Points to the block containing the
           -- file information.

    item : IN element_type;
           -- Holds the data being written.

    to   : IN positive_count
           -- Index of the element in the file to be used for
           -- transfer.

   );


PROCEDURE WRITE
-- will write to a direct access file at the position
-- following the last written position.
   (file : IN file_type;
           -- Points to the block containing the
           -- file information.

    item : IN element_type
```

```
                  -- Holds the data being written.

         );


      PROCEDURE SET_INDEX
      -- will set a direct access file's index to the given
      -- position.
         (file : IN file_type;
                  -- Points to the block containing the
                  -- file information.

          to   : IN positive_count
                  -- Specifies the new index to be set.

         );


      FUNCTION INDEX
      -- gives the position of a direct access file's index.
         (file : IN file_type
                  -- Points to the block containing the
                  -- file information.

         ) RETURN positive_count;
                   -- The current index position.


      FUNCTION SIZE
      -- gives the size of the direct access file.
         (file  : IN file_type
                   -- Points to the block containing the
                   -- file information.

         ) RETURN count;
                   -- Size in elements of the file.


      FUNCTION END_OF_FILE
      -- indicates whether the end of the direct access file has
      -- been reached.
         (file  : IN file_type
                   -- Points to the block containing the
                   -- file information.

         ) RETURN BOOLEAN;
                   -- Indicator of end of file test.

   PRAGMA PAGE;    -- In Package DIRECT_IO Specification

   -- The following are the I/O-specific Ada exceptions.

   status_error : EXCEPTION RENAMES io_exceptions.status_error;
                   -- Indicates that the file is not properly set
```

```
                    -- for the requested operation.

mode_error    : EXCEPTION RENAMES io_exceptions.mode_error;
                -- Indicates that the file was not in the proper
                -- mode for an attempted read or write operation.

name_error    : EXCEPTION RENAMES io_exceptions.name_error;
                -- Indicates that the syntax of the name is not
                -- valid or that the name has already been used.

use_error     : EXCEPTION RENAMES io_exceptions.use_error;
                -- Indicates the improper usage of a file.

device_error : EXCEPTION RENAMES io_exceptions.device_error;
                -- Indicates that a device driver program has
                -- failed for one reason or another.

end_error     : EXCEPTION RENAMES io_exceptions.end_error;
                -- Indicates that an attempt was made to read
                -- past the end of the file.

data_error    : EXCEPTION RENAMES io_exceptions.data_error;
                -- Indicates that an attempt was made, to read
                -- from a file into a buffer or write from a
                -- buffer into a file, data of the wrong type.

PRAGMA PAGE;    -- In Package DIRECT_IO Specification

-- The following are the private implementation-dependent
-- declarations.

PRIVATE

TYPE file_type IS
   -- Contains all the needed information on the file and its
   -- properties of concern to the I/O operation.
   RECORD
      real_file_type : io_support.file_type;
                       -- Contains the needed information for
                       -- direct reading and/or writing of the
                       -- data.
   END RECORD;

END DIRECT_IO;
```

```
*******************************************************************
PACKAGE SEQUENTIAL_IO

-- C 1987 United States Government as represented by
--    the Secretary of the Navy. ALL RIGHTS RESERVED.

--    (The U.S. Government possesses the unlimited rights
--    throughout the world for government purposes to
--    publish, translate, reproduce, deliver, perform, and
--    dispose of the technical data, computer software, or
--    computer firmware contained herein; and to authorize
--    others to do so.)

--
--
-- Revision History:

--
--      3 Feb 1987  JGR
--          Package Specification Created.


WITH IO_EXCEPTIONS, IO_SUPPORT;

GENERIC

   TYPE element_type IS PRIVATE;
      -- The type of the element instantiated on the call to any
      -- of the subprograms of this package.  It is determined
      -- by the type of the element being used in the invoking
      -- subprogram.

PACKAGE SEQUENTIAL_IO IS

PRAGMA PAGE;    -- In Package SEQUENTIAL_IO Specification

--| JUSTIFICATION:
--|
--|     SEQUENTIAL_IO contains the subprograms necessary for the
--| user to perform sequential access I/O operations (as
--| described in Section 14.2.1 and Section 14.2.3 of
--| [ANSI/MIL-STD-1815]).  SEQUENTIAL_IO is primarily an
--| interface, between the User-Written_Ada_Program and the
--| package IO_SUPPORT which does the actual work for
--| SEQUENTIAL_IO operations. In serving as this interface,
--| SEQUENTIAL_IO contains only the calls necessary to invoke
--| the subprograms of IO_SUPPORT. SEQUENTIAL_IO is a generic
--| package thus enabling it to work for all instantiated types.
--|
--|     SEQUENTIAL_IO also will make use of the Ada feature to
--| overload subprogram names.  In the cases where overloading
--| is used, each subprogram will be listed separately in its
--| entirety.
--| DATA:
--|
--|     Type declarations for the limited private type FILE_TYPE
--| and the type FILE_MODE are declared in this package
--| specification.


TYPE file_type IS LIMITED PRIVATE;
   -- Forward declaration.  The formal declaration
   -- will appear later.

TYPE file_mode IS
   -- Allowable modes for sequential files.
   (in_file,   -- Input mode (read only).
    out_file   -- Output mode (write only).
   );


--| INITIALIZATION:
--|
--|     None.


--| CREATED TASKS:
--|
--|     None.


--| SUBPROGRAMS AND TASKS:
--|
--|

   PROCEDURE CREATE
   -- will create a file for sequential access.
      (file : IN OUT file_type;
              -- Points to the block containing the file
              -- information.

       mode : IN file_mode := out_file;
              -- Specifies that the file is to be written to.

       name : IN STRING := "";
              -- The name of the external file.

       form : IN STRING := ""
              -- Used by the system for file characteristics.

      );


   PROCEDURE OPEN
   -- will open a file for sequential access.
      (file : IN OUT file_type;
              -- Points to the block containing the
              -- file information.
```

```
     mode : IN file_mode;
            -- Specifies whether the file is to be read from or
            -- written to.

     name : IN STRING;
            -- The name of the external file.

     form : IN STRING := ""
            -- Used by the system for file characteristics.

    );


PROCEDURE CLOSE
-- will close a sequential access file.
    (file : IN OUT file_type
            -- Points to the block containing the
            -- file information.

    );


PROCEDURE DELETE
-- will deiete a sequential access file.
    (file : IN OUT file_type
            -- Points to the block containing the
            -- file information.

    );


PROCEDURE RESET
-- will reset a sequential access file and change its mode
-- to the requested mode.
    (file : IN OUT file_type;
            -- Points to the block containing the
            -- file information.

     mode : IN file_mode
            -- Specifies whether the file is to be read from or
            -- written to.

    );


PROCEDURE RESET
-- will reset a sequential access file but will not change
-- its mode.
    (file : IN OUT file_type
            -- Points to the block containing the
            -- file information.

    );
```

```
FUNCTION MODE
-- gives the mode of the sequential access file.
   (file  : IN file_type
            -- Points to the block containing the
            -- file information.

   ) RETURN file_mode;
            -- Present mode of the given file.


FUNCTION NAME
-- gives the external name of the sequential access file.
   (file  : IN file_type
            -- Points to the block containing the
            -- file information.

   ) RETURN STRING;
            -- Full name of the external file.


FUNCTION FORM
-- gives the form of the sequential access file.
   (file  : IN file_type
            -- Points to the block containing the
            -- file information.

   ) RETURN STRING;
            -- Form (system file characteristics) of the
            -- given file.


FUNCTION IS_OPEN
-- indicates whether the sequential access file is open.
   (file  : IN file_type
            -- Points to the block containing the
            -- file information.

   ) RETURN BOOLEAN;
            -- Whether or not the file is open.


PROCEDURE READ
-- will read from a sequential access file.
   (file : IN file_type;
           -- Points to the block containing the
           -- file information.

    item : OUT element_type
           -- Holds the data being read.

   );


PROCEDURE WRITE
```

```
   -- will write to a sequential access file.
      (file : IN file_type;
              -- Points to the block containing the
              -- file information.

       item : IN element_type
              -- Holds the data to be written.

      );


   FUNCTION END_OF_FILE
   -- indicates whether the end of the sequential access file
   -- has been reached.
      (file  : IN file_type
               -- Points to the block containing the
               -- file information.

      ) RETURN BOOLEAN;
               -- Result of the end of file test.

PRAGMA PAGE;   -- In Package SEQUENTIAL_IO Specification

-- The following are the I/O-specific Ada exceptions.

status_error : EXCEPTION RENAMES io_exceptions.status_error;
               -- Indicates that the file is not properly set
               -- for the requested operation.

mode_error   : EXCEPTION RENAMES io_exceptions.mode_error;
               -- Indicates that the file was not in the proper
               -- mode for an attempted read or write operation.

name_error   : EXCEPTION RENAMES io_exceptions.name_error;
               -- Indicates that the syntax of the name is not
               -- valid or that the name has already been used.

use_error    : EXCEPTION RENAMES io_exceptions.use_error;
               -- Indicates the improper usage of a file.

device_error : EXCEPTION RENAMES io_exceptions.device_error;
               -- Indicates that a device driver program has
               -- failed for one reason or another.

end_error    : EXCEPTION RENAMES io_exceptions.end_error;
               -- Indicates that an attempt was made to read
               -- past the end of the file.

data_error   : EXCEPTION RENAMES io_exceptions.data_error;
               -- Indicates that an attempt was made, to read
               -- from a file into a buffer or write from a
               -- buffer into a file, data of the wrong type.

PRAGMA PAGE;   -- In Package SEQUENTIAL_IO Specification
```

```
-- The following is the private implementation-dependent
-- declaration.

PRIVATE

TYPE file_type IS
   -- Contains all the needed information on the file and its
   -- properties of concern to the I/O operation.
   RECORD
      real_file_type : io_support.file_type;
                       -- Contains the needed information for
                       -- sequential reading and/or writing of
                       -- the data.
   END RECORD;

END SEQUENTIAL_IO;
```

```
*********************************************************************
PACKAGE IO_DEFS

-- C 1987 United States Government as represented
-- by the Secretary of the Navy. ALL RIGHTS RESERVED.

--   (The U.S. Government possesses the unlimited rights
--   throughout the world for government purposes to
--   publish, translate, reproduce, deliver, perform, and
--   dispose of the technical data, computer software, or
--   computer firmware contained herein; and to authorize
--   others to do so )

--
--
-- Revision History:

--
--     3 Feb 1987  JGR
--         Package Specification Created.

WITH SYSTEM;

PACKAGE IO_DEFS IS

PRAGMA PAGE;   -- In Package IO_DEFS Specification

--| JUSTIFICATION:
--|
--|     IO_DEFS contains the implementation-dependent object and
--| type declarations used in I/O_Management/RTLib. Hence,
--| IO_DEFS allows the I/O_Management/RTLib packages to stay
--| implementation-independent.


--| DATA:
--|
--|     The data in this package will consists of CONSTANTS,
--| TYPES and SUBTYPES. The data will define the
--| implementation-dependent objects and types, such as
--| file lengths, data size, transfer limits, buffer limits,
--| file modes and accessibility. The peripheral devices
--| supported will also be contained in an enumerication type.


io_device_map_size   : CONSTANT := 32;
                       -- I/O_Device_Map array size.

data_maxln_k         : CONSTANT := 2_147_483_647;
                       -- The largest positive integer on the
                       -- AN/UYK(43).

file_minln_k         : CONSTANT := -2_147_483_647;
                       -- The smallest negative integer on
                       -- the AN/UYK(43).
```

```
file_maxln_k        : CONSTANT := 2_147_483_647;
                      -- The largest positive integer on the
                      -- AN/UYK(43).

ext_name_length     : CONSTANT := 20;
                      -- The limit of the string length for
                      -- holding an external file name.

max_form_length     : CONSTANT := 20;
                      -- The limit of the form length for
                      -- holding the file creation form parameter.

buffer_data_limit   : CONSTANT := 256*4;
                      -- This is the maximum amount of data to
                      -- be put into a text data buffer to hand to
                      -- the Ada/L device drivers to transfer.


SUBTYPE channel_range_int IS INTEGER
                          RANGE 0..63;
                          -- This defines the allowable range
                          -- of values used in the determination
                          -- of which I/O channel is to be
                          -- accessed.

SUBTYPE unit_range_int  IS INTEGER
                          RANGE 0..9;
                          -- This defines the allowable range
                          -- of values used in the determination
                          -- of which unit on a device is to
                          -- be used.

SUBTYPE data_length_int IS INTEGER
                          RANGE 0..data_maxln_k;
                          -- The allowable range of maximum
                          -- values for a file record element.
                          -- Also the last position in all buffers,
                          -- lines, columns, and pages, within
                          -- package TEXT_IO.  Zero here
                          -- indicates the special case that
                          -- the item being indexed is empty.

SUBTYPE file_length_int IS INTEGER
                          RANGE 0..file_maxln_k;
                          -- The allowable range of maximum
                          -- values for an index used to mark
                          -- the position in a file.  Zero
                          -- here indicates the special case
                          -- that the item being indexed is
                          -- empty.

SUBTYPE device_range_int IS INTEGER
                          RANGE 1..io_defs.io_device_map_size;
                          -- Type for I/O_Device_Map index
```

```
                                   -- (device_id).

SUBTYPE file_name_str      IS STRING(1..io_defs.ext_name_length);
                                   -- The external name of a file.

SUBTYPE form_str          IS STRING(1..io_defs.max_form_length);
                                   -- The form specified at file creation.

blank_file_name : CONSTANT file_name_str := (OTHERS => ' ');
                         -- A blank file name used for initialization.

SUBTYPE device_mnemonics_type IS STRING(1..2);
   -- A two character device mnemonic.

TYPE status_type IS
   -- A flag used for device and channel status.
   (up,                         -- used to set status to up.
    down,                       -- used to set status to down.
    none                        -- used when no secondary channel
                                -- is supplied
   );

TYPE channel_type_enu IS
   -- The allowable channel types.
   (computer_device_16,         -- computer to device,
                                -- 16 data bits.
    computer_computer_16,       -- computer to computer,
                                -- 16 data bits.
    computer_device_32,         -- computer to device,
                                -- 32 data bits.
    computer_computer_32        -- computer to computer,
                                -- 32 data bits.
   );

TYPE io_mode_enu IS
   -- Defines the type of I/O mode for the file referenced.
   (iom_in,                     -- Read mode, an input operation.
    iom_out,                    -- Write mode, an output operation.
    iom_inout                   -- Read and write mode, in input or
                                -- output operation.
   );

TYPE accessibility_enu IS
   -- Defines a corresponding value for the allowable types of
   -- I/O access.  These values are of particular importance to
   -- the device drivers to indicate what type of read/write
   -- operations will be expected of any one driver.
   (a_sequential,               -- Read/Write access will be
                                -- sequential.
    a_direct,                   -- Read/Write access will be
                                -- direct.
    a_both                      -- Read/Write access will be
                                -- sequential or direct.
   );
```

```
TYPE file_class_enu IS
   -- Defines the allowable predefined types of alpha-numeric
   -- contents for a file in the Ada/L I/O.
   (fc_data,                  -- The file contents may be data
                              -- only.
    fc_text                   -- The file contents may be text
                              -- only.
   );

TYPE io_result_enu IS
   -- Defines the status of the requested I/O operation
   -- immediately following its return.  The variable, local to
   -- the subprogram that requested the I/O operation (the
   -- invoking subprogram), of this type should be checked
   -- directly after the return from the I/O operation (the
   -- invoked subprogram) for its I/O result enumeration status.
   -- Hence, this will allow the raising of exceptions at the
   -- needed points in the execution.
   (ior_ok,                  -- I/O request executed
                             -- satisfactorily.
    ior_end_info,            -- I/O request reached the
                             -- end of file.
    ior_access_err,          -- I/O request made an
                             -- invalid access.
    ior_name_err,            -- I/O request had a name
                             -- syntax error.
    ior_device_err,          -- I/O request had a
                             -- system malfunction.
    ior_storage_err,         -- I/O request accessed
                             -- storage wrong.
    ior_data_err,            -- I/O request contained
                             -- invalid data.
    ior_operation_err,       -- I/O request operation
                             -- failed.
    ior_node_exists,         -- I/O request reached
                             -- a valid node.
    ior_node_doesnt_exist,   -- I/O request reached
                             -- an invalid node.
    ior_default_chosen       -- I/O request using the
                             -- default device.
   );

TYPE peripheral_device_enu IS
   -- Used to specify which device the external file is on.
   -- NOTE_1: The enumeration values "user_device_?" are for
   --         user written and added device driver tasks.  The
   --         current design in the Ada/L I/O allows for the
   --         addition of six device driver tasks.
   (usq69_device,            -- Specifies the USQ-69 display
                             -- terminal.
    uyh3_device,             -- Specifies the UYH-3 disk drive.
    rd358_device,            -- Specifies the RD-358 tape drive.
    milstd1397_interface,    -- Specifies a MilStd-1397A computer
                             -- interface.
    milstd1553_interface,    -- Specifies a MilStd-1553B computer
```

```
                                 -- interface.
     user_device_a,              -- User-defined device type A.
     user_device_b,              -- User-defined device type B.
     user_device_c,              -- User-defined device type C.
     user_device_d,              -- User-defined device type D.
     user_device_e,              -- User-defined device type E.
     user_device_f,              -- User-defined device type F.
     user_device_g,              -- User-defined device type G.
     user_device_h,              -- User-defined device type H.
     user_device_i,              -- User-defined device type I.
     user_device_j,              -- User-defined device type J.
     user_device_k,              -- User-defined device type K.
     user_device_l,              -- User-defined device type L.
     user_device_m,              -- User-defined device type M.
     user_device_n,              -- User-defined device type N.
     user_device_o,              -- User-defined device type O.
     user_device_p               -- User-defined device type P.
    );


 TYPE io_operations_enu IS
    -- Identifies the different I/O operations supported by the
    -- Ada/L I/O.  These operations are an intricate part of the
    -- File_Info_Block used by the System_I/O level code.
    -- IO_MANAGEMENT/RTLIB layers FILE_IO and PHYSICAL_IO
    -- will make explicit use of these codes.
    (create_request,             -- Create a new external file.
     open_request,               -- Open an already existing file.
     close_request,              -- Close the specified file.
     delete_request,             -- Delete the specified file.
     read_request,               -- Read a record from the specified
                                 -- file.
     write_request,              -- Write a record to the specified
                                 -- file.
     set_index_request,          -- Set a specified position in the
                                 -- file.
     size_request,               -- Determines the number of records
                                 -- in the file.
     reset_request,              -- Reset the specified file.
     eof_request                 -- Determines the end of file
                                 -- boolean.
    );


 TYPE io_request_block IS
    -- This structure defines the I/O_Request_Block used by the
    -- device drivers to service all I/O requests.  The data,
    -- discriminanted by the peripheral type, contained in this
    -- record is the information required for the proper
    -- communication between FILE_IO and PHYSICAL_IO.
    RECORD
       function_request : INTEGER;
                              -- The device specific function request.
                              -- Each device driver will interpret
                              -- the contents of this integer by
```

```
                                  -- associating it with an enumeration
                                  -- type which is internal to the device
                                 --- driver task body.

      device_type               : io_defs.peripheral_device_enu;
                                  -- Indicates the device type
                                  -- that the io request is being
                                  -- directed to.

      device_id                 : io_defs.device_range_int;
                                  -- Identifies the index into the
                                  -- I/O_Device_Map.

      unit_number               : io_defs.unit_range_int;
                                  -- Indicates the specific unit number
                                  -- to use on the peripheral devices.

      data_location             : system.address;
                                  -- Indicates the address of the
                                  -- data buffer.

      data_length               : io_defs.data_length_int;
                                  -- Indicates the number of words
                                  -- in data buffer.

      status                    : io_defs.io_result_enu;
                                  -- Indicates the resulting status of
                                  -- I/O operation.

      ei_word                   : INTEGER;
                                  -- The returned external interrupt
                                  -- word.

      filler1,
      filler2,
      filler3,
      filler4,
      filler5                   : INTEGER;
                                  -- Used to pass device specific info,
                                  -- the contents will very by device
                                  -- type.
                                  -- NOTE: For DISK_IO these fillers are
                                  -- used as specified below:
                                  -- Filler1 := disk_cylinder,
                                  -- Filler2 := disk_sector,
                                  -- Filler3 := disk_head,
                                  -- Filler4/5 := not used.
   END RECORD;


TYPE iorb_access IS ACCESS io_request_block;
   -- Provides access for allocating an IO_Request_Block.

--| INITIALIZATION:
--|
--|     All constant data will be initialized to an initial
--| value representing a implementation-dependent fixed value.


--| CREATED TASKS:
--|
--|     None.


--| SUBPROGRAMS AND TASKS:
--|
--|     None.

END IO_DEFS;
```

```
******************************************************************************
PACKAGE STANDARD

The package STANDARD contains the following definitions in
addition to those specified in Appendix C of ANSI/MIL-STD-1815A.

--      c 1987 United States Government as represented by
--        the Secretary of Navy. ALL RIGHTS RESERVED.
--
--        (The U.S. Government possesses the unlimited rights
--        throughout the world for Government purposes to
--        publish, translate, reproduce, deliver, perform, and
--        dispose of the technical data, computer software, or
--        computer firmware contained herein; and to authorize
--        others to do so.)
--   REVISION HISTORY:
--   10 Mar 1987 TCJ
--      Coded from PDL

PACKAGE STANDARD IS
--| JUSTIFICATION:
--|
--|     STANDARD contains the definitions for the pre-defined
--| types and constants as defined in Appendix C of
--| ANSI/MIL-STD-1815A.
--|
--| Assumptions:
--|
--|     STANDARD is targeted for Ada/L(43).
--| TYPES and DATA:
--|
--|     See below.
   TYPE boolean IS (false, true);
-- FOR  boolean'SIZE USE 1;
-- The predefined relational operators for this type are as
-- follows:
-- FUNCTION "="  (left, right : boolean) RETURN boolean;
-- FUNCTION "/=" (left, right : boolean) RETURN boolean;
-- FUNCTION "<"  (left, right : boolean) RETURN boolean;
-- FUNCTION "<=" (left, right : boolean) RETURN boolean;
-- FUNCTION ">"  (left, right : boolean) RETURN boolean;
-- FUNCTION ">=" (left, right : boolean) RETURN boolean;
-- The predefined logical operators and the predefined logical
-- negation operators are as follows:
--    FUNCTION "AND" (left, right : boolean) RETURN boolean;
--    FUNCTION "OR"  (left, right : boolean) RETURN boolean;
--    FUNCTION "XOR" (left, right : boolean) RETURN boolean;
--    FUNCTION "NOT" (right : boolean) RETURN boolean;
-- The type universal_integer is predefined.
   TYPE integer IS RANGE -2_147_483_647 .. 2_147_483_647;
-- -(2**31 - 1) .. (2**31 - 1)
-- FOR  integer'SIZE USE 32;
-- The predefined operators for this type are as follows:
--      FUNCTION "="   (left, right : integer) RETURN boolean;
--      FUNCTION "/="  (left, right : integer) RETURN boolean;
--     FUNCTION "<"   (left, right : integer) RETURN boolean;
--     FUNCTION "<="  (left, right : integer) RETURN boolean;
--     FUNCTION ">"   (left, right : integer) RETURN boolean;
--     FUNCTION ">="  (left, right : integer) RETURN boolean;
--     FUNCTION "+"   (right : integer) RETURN integer;
--     FUNCTION "-"   (right : integer) RETURN integer;
--     FUNCTION "abs" (right : integer) RETURN integer;
--     FUNCTION "+"   (left, right : integer) RETURN integer;
--     FUNCTION "-"   (left, right : integer) RETURN integer;
--     FUNCTION "*"   (left, right : integer) RETURN integer;
--     FUNCTION "/"   (left, right : integer) RETURN integer;
--     FUNCTION "rem" (left, right : integer) RETURN integer;
--     FUNCTION "mod" (left, right : integer) RETURN integer;
--     FUNCTION "**"  (left : integer; right : integer) RETURN
--         integer;
   TYPE long_integer IS RANGE
      -9_223_372_036_854_775_807 .. 9_223_372_036_854_775_807;
-- The predefined operators for this type are as follows:
--     FUNCTION "="   (left, right : long_integer) RETURN boolean;
--     FUNCTION "/="  (left, right : long_integer) RETURN boolean;
--     FUNCTION "<"   (left, right : long_integer) RETURN boolean;
--     FUNCTION "<="  (left, right : long_integer) RETURN boolean;
--     FUNCTION ">"   (left, right : long_integer) RETURN boolean;
--     FUNCTION ">="  (left, right : long_integer) RETURN
--         boolean;
--     FUNCTION "+"   (right : long_integer) RETURN long_integer;
--     FUNCTION "-"   (right : long_integer) RETURN long_integer;
--     FUNCTION "abs" (right : long_integer) RETURN long_integer;
--     FUNCTION "+"   (left, right : long_integer) RETURN
--         long_integer;
--     FUNCTION "-"   (left, right : long_integer) RETURN
--         long_integer;
--     FUNCTION "*"   (left, right : long_integer) RETURN
--         long_integer;
--     FUNCTION "/"   (left, right : long_integer) RETURN
--         long_integer;
--     FUNCTION "rem" (left, right : long_integer) RETURN
--         long_integer;
--     FUNCTION "mod" (left, right : long_integer) RETURN
--         long_integer;
--     FUNCTION "**"  (left : long_integer; right : integer) RETURN
--         long_integer;
   TYPE float IS DIGITS 6 RANGE
                -(16#0.FF_FFFF#E63) ..
                 (16#0.FF_FFFF#E63);
-- The predefined operators for this type are as follows:
--     FUNCTION "="   (left, right : float) RETURN boolean;
--     FUNCTION "/="  (left, right : float) RETURN boolean;
--     FUNCTION "<"   (left, right : float) RETURN boolean;
--     FUNCTION "<="  (left, right : float) RETURN boolean;
--     FUNCTION ">"   (left, right : float) RETURN boolean;
--     FUNCTION ">="  (left, right : float) RETURN boolean;
--     FUNCTION "+"   (right : float) RETURN float;
--     FUNCTION "-"   (right : float) RETURN float;
--     FUNCTION "abs" (right : float) RETURN float;
--     FUNCTION "+"   (left, right : float) RETURN float;
--     FUNCTION "-"   (left, right : float) RETURN float;
--     FUNCTION "*"   (left, right : float) RETURN float;
--     FUNCTION "/"   (left, right : float) RETURN float;
--     FUNCTION "**"  (left : float; right : integer) RETURN float;
   TYPE long_float IS DIGITS 15 RANGE
                  -(16#0.FF_FFFF_FFFF_FFFF#E63) ..
                   (16#0.FF_FFFF_FFFF_FFFF#E63);
-- The predefined operators for this type are as follows:
--     FUNCTION "="   (left, right : long_float) RETURN boolean;
--     FUNCTION "/="  (left, right : long_float) RETURN boolean;
--     FUNCTION "<"   (left, right : long_float) RETURN boolean;
--     FUNCTION "<="  (left, right : long_float) RETURN boolean;
--     FUNCTION ">"   (left, right : long_float) RETURN boolean;
--     FUNCTION ">="  (left, right : long_float) RETURN boolean;
--     FUNCTION "+"   (right : long_float) RETURN long_float;
--     FUNCTION "-"   (right : long_float) RETURN long_float;
--     FUNCTION "abs" (right : long_float) RETURN long_float;
--     FUNCTION "+"   (left, right : long_float) RETURN long_float;
--     FUNCTION "-"   (left, right : long_float) RETURN long_float;
--     FUNCTION "*"   (left, right : long_float) RETURN long_float;
--     FUNCTION "/"   (left, right : long_float) RETURN long_float;
--     FUNCTION "**"  (left : long_float; right : integer) RETURN
--          long_float;
-- In addition, the following operators are predefined for universal
-- types:
--     FUNCTION "*"   (left : universal_integer;
--                     right : universal_real)
--                     RETURN universal_real;
--
--     FUNCTION "*"   (left : universal_real;
--                     right : universal_integer)
--                     RETURN universal_real;
--
--     FUNCTION "/"   (left : universal_real;
--                     right : universal_integer)
--                     RETURN universal_real;
-- The type universal_fixed is predefined. The only operators
-- declared for this type are:

--     FUNCTION "*"  (left:  any_fixed_point_type;
--                    right : any_fixed_point_type)
--                       RETURN universal_fixed;
--     FUNCTION "/"  (left :  any_fixed_point_type;
--                    right : any_fixed_point_type)
--                       RETURN universal_fixed;

-- The following characters form the standard ASCII set.

  TYPE character IS
      ( -- 32 control characters are defined here --

        ' ', '!', '"', '#',    '$', '%', '&', ''',
        '(', ')', '*', '+',    ',', '-', '.', '/',
        '0', '1', '2', '3',    '4', '5', '6', '7',
        '8', '9', ':', ';',     '<', '=', '>', '?',

        '@', 'A', 'B', 'C',     'D', 'E', 'F', 'G',
        'H', 'I', 'J', 'K',     'L', 'M', 'N', 'O',
        'P', 'Q', 'R', 'S',     'T', 'U', 'V', 'W',
        'X', 'Y', 'Z', '[',     '\', ']', '^', '_',

        '`', 'a', 'b', 'c',     'd', 'e', 'f', 'g',
        'h', 'i', 'j', 'k',     'l', 'm', 'n', 'o',
        'p', 'q', 'r', 's',     't', 'u', 'v', 'w',
        'x', 'y', 'z', '{',     '|', '}', '~', '');

   FOR character USE  -- 128 ASCII character set without holes
      ( 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,
       10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
       20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
       30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
       40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
       50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
       60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
       70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
       80, 81, 82, 83, 84, 85, 86, 87, 88, 89,
       90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
      100,101,102,103,104,105,106,107,108,109,
      110,111,112,113,114,115,116,117,118,119,
      120,121,122,123,124,125,126,127 );

-- FOR character'SIZE USE 8;

   -- The predefined operators for the type CHARACTER are the same
   -- as for any enumeration type.

   PACKAGE ASCII IS

      -- Control characters:

      nul : CONSTANT character := ' ';
      soh : CONSTANT character := ' ';
      stx : CONSTANT character := ' ';
      etx : CONSTANT character := ' ';
      eot : CONSTANT character := ' ';
      enq : CONSTANT character := ' ';
      ack : CONSTANT character := ' ';
      bel : CONSTANT character := ' ';
      bs  : CONSTANT character := ' ';
      ht  : CONSTANT character := '      ';
      lf  : CONSTANT character := ' ';
      vt  : CONSTANT character := ' ';
      ff  : CONSTANT character := ' ';
      cr  : CONSTANT character := ' ';
      so  : CONSTANT character := ' ';
      si  : CONSTANT character := ' ';
      dle : CONSTANT character := ' ';
      dc1 : CONSTANT character := ' ';
      dc2 : CONSTANT character := ' ';
```

```
dc3 : CONSTANT character := ' ';
dc4 : CONSTANT character := ' ';
nak : CONSTANT character := ' ';
syn : CONSTANT character := ' ';
etb : CONSTANT character := ' ';
can : CONSTANT character := ' ';
em  : CONSTANT character := ' ';
sub : CONSTANT character := ' ';
esc : CONSTANT character := ' ';
fs  : CONSTANT character := ' ';
gs  : CONSTANT character := ' ';
rs  : CONSTANT character := ' ';
us  : CONSTANT character := ' ';
del : CONSTANT character := ' ';

-- Other characters:

exclam     : CONSTANT character := '!';
sharp      : CONSTANT character := '#';
dollar     : CONSTANT character := '$';
query      : CONSTANT character := '?';
at_sign    : CONSTANT character := '@';
l_bracket  : CONSTANT character := '[';
back_slash : CONSTANT character := '\';
r_bracket  : CONSTANT character := ']';
circumflex : CONSTANT character := '^';
grave      : CONSTANT character := '`';
l_brace    : CONSTANT character := '{';
bar        : CONSTANT character := '|';
r_brace    : CONSTANT character := '}';
tilde      : CONSTANT character := '~';

-- Lower case letters:

lc_a : CONSTANT character := 'a';
lc_b : CONSTANT character := 'b';
lc_c : CONSTANT character := 'c';
lc_d : CONSTANT character := 'd';
lc_e : CONSTANT character := 'e';
lc_f : CONSTANT character := 'f';
lc_g : CONSTANT character := 'g';
lc_h : CONSTANT character := 'h';
lc_i : CONSTANT character := 'i';
lc_j : CONSTANT character := 'j';
lc_k : CONSTANT character := 'k';
lc_l : CONSTANT character := 'l';
lc_m : CONSTANT character := 'm';
lc_n : CONSTANT character := 'n';
lc_o : CONSTANT character := 'o';
lc_p : CONSTANT character := 'p';
lc_q : CONSTANT character := 'q';
lc_r : CONSTANT character := 'r';
lc_s : CONSTANT character := 's';
lc_t : CONSTANT character := 't';
lc_u : CONSTANT character := 'u';
```

```
      lc_v : CONSTANT character := 'v';
      lc_w : CONSTANT character := 'w';
      lc_x : CONSTANT character := 'x';
      lc_y : CONSTANT character := 'y';
      lc_z : CONSTANT character := 'z';

   END ASCII;
   -- Predefined subtypes:

   SUBTYPE natural  IS integer RANGE 0 .. integer'LAST;
   SUBTYPE positive IS integer RANGE 1 .. integer'LAST;

   -- Predefined string type:

   TYPE string IS ARRAY (positive RANGE <>) OF character;

   PRAGMA PACK(string);
   -- The predefined operators for this type are as follows:

   -- FUNCTION "="  (left, right : string) RETURN boolean;
   -- FUNCTION "/=" (left, right : string) RETURN boolean;
   -- FUNCTION "<"  (left, right : string) RETURN boolean;
   -- FUNCTION "<=" (left, right : string) RETURN boolean;
   -- FUNCTION ">"  (left, right : string) RETURN boolean;
   -- FUNCTION ">=" (left, right : string) RETURN boolean;

   -- FUNCTION "&"  (left : string;    right : string)
   --                RETURN boolean;
   -- FUNCTION "&"  (left : character; right : string)
   --                RETURN boolean;
   -- FUNCTION "&"  (left : string;    right : character)
   --                RETURN boolean;
   -- FUNCTION "&"  (left : character; right : character)
   --                RETURN boolean;

   TYPE duration IS DELTA 2.0 **(-14) RANGE -131_071.0..131_071.0;
   --                                     -- (2**17 - 1)

   -- The predefined operators for the type DURATION are the same
   -- as for any fixed point type.
   -- The predefined exceptions:

   constraint_error : exception;
   numeric_error    : exception;
   program_error    : exception;
   storage_error    : exception;
   tasking_error    : exception;

END STANDARD;
```

APPENDIX C

TEST PARAMETERS

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.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	<1..119 => 'A', 120 => '1'>
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	<1..119 => 'A', 120 => '2'>
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	<1..59 => 'A', 60 => '3', 61..120 => 'A'>
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	<1..59 => 'A', 60 => '4', 61..120 => 'A'>
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	<1..117 => '0', 118..120 => '298'>
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	<1..115 => '0', 116..120 => '690.0'>

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	<1 => '"', 2..61 => 'A', 62 => '"'>
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	<1 => '"', 2..60 => 'A', 61 => '1', 62 => '"'>
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	<1..100 => ' '>
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	2143483647
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	BAD-CHARS^#-TOO-LONG-A-FILE-NAME.%!X
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	WILD-CHAR*-TOO-LONG-A-FILE-NAME.NAM
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	131_071.5
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131_073.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BADCHAR^@-TOO-LONG-A-FILE-NAME.~!

$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	MUCH_TOO_LONG_NAME_FOR_A_FILE
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2147483647
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2147483648
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-131_071.5
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131_073.0
$MAX_DIGITS Maximum digits supported for floating-point types.	6
$MAX_IN_LEN Maximum input line length permitted by the implementation.	120
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	9223372036854775807
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	9223372036854775808
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	<1..2 => '2:', 3..117 => '0', 118..120 => '11:'>

$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.	<1..3 => '16:', 4..116 => '0', 117..120 => 'F.E:'>
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	<1 => '"', 2..119 => 'A', 120 => '"'>
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_ INT.	-9223372036854775807
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	No_Such_Type
$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.	16#FFFFFFFFFFFFFFFD#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 28 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A: A basic declaration (line 36) wrongly follows a later declaration.

E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.

C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.

C35502P: Equality operators in lines 62 & 69 should be inequality operators.

A35902C: Line 17's assignment of the nominal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

C35A03E, & R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.

C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

C37215C, E, G, H:	Various discriminant constraints are wrongly expected to be incompatible with type CONS.
C38102C:	The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.
C41402A:	'STORAGE_SIZE is wrongly applied to an object of an access type.
C45332A:	The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.
C45614C:	REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).
E66001D:	This test wrongly allows either the acceptance or rejection of a parameterless function with the same identifier as an enumeration literal; the function must be rejected (see Commentary AI-00330).
A74106C, C85018B, C87B04B, CC1311B:	A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively (and possibly elsewhere).
BC3105A:	Lines 159..168 are wrongly expected to be illegal; they are legal.
AD1A01A:	The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.
CE2401H:	The record aggregates in lines 105 & 117 contain the wrong values.
CE3208A:	This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.