REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

Public re... needed, ... Headqu... Manage...

AD-A250 163

...er response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data ...ian estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington ...on Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of

1. AGE...	DATE	3. REPORT TYPE AND DATES COVERED
		Final: 30 Nov 1990 to 01 Jun 1993

4. TITLE AND SUBTITLE
Validation Summary Report: Concurrent Computer Corporation, C3 Ada Version R03-00V, Concurrent Computer Corporation 3280MP under OS/32 Version R08-03.2 (Host & Target), 901130W1.11108

5. FUNDING NUMBERS

DTIC ELECTE MAY 5 1992 S C D

6. AUTHOR(S)
Wright-Patterson AFB, Dayton, OH
USA

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

8. PERFORMING ORGANIZATION REPORT NUMBER
AVF-VSR-415.0891

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

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
Concurrent Computer Corporation, C3 Ada Version R03-00V, Wright-Patterson, AFB, Concurrent Computer Corporation 3280MP under OS/32 Version R08-03.2 (Host & Target), ACVC 1.11.

92 4 29 075

92-11777

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

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFED	UNCLASSIFIED	

NSN 7540-01-280-550

Standard Form 298, (Rev. 2-89)
Prescribed by ANSI Std 239-128

AVF Control Number: AVF-VSR-415.0891
1 August 1991
90-10-08-CCC

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 901130W1.11108
Concurrent Computer Corporation
C3 Ada Version R03-00V
Concurrent Computer Corporation 3280MPS under OS/32 Version R08-03.2
(self-targeted)

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

DECLARATION OF CONFORMANCE

Customer:	Concurrent Computer Corporation
Ada Validation Facility:	Wright Patterson Air Force Base, Ohio.
ACVC Version:	1.11

Ada Implementation:

Compiler Name and Version:	C ^{3}Ada **Version:** R03-00V
Host Computer System:	Concurrent Computer Corporation 3280 MPS under OS/32 Version R08-03.2
Target Computer System:	Same as Host

Customer's Declaration

I, the undersigned, representing Concurrent Computer Corporation, declare that Concurrent Computer Corporation has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration. I declare that Concurrent Computer Corporation is the Implementor of the above implementation and the certificates shall be awarded in the name of Concurrent Computer Corporation's corporate name.

Seetharama Shastry 11/29/90

Seetharama Shastry (date)
Senior Manager, System Software Development

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

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

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

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

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

1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

1.3 ACVC TEST CLASSES

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

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

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

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values — for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1) and, possibly some inapplicable tests (see Section 2.2 and [UG89]).

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

1.4 DEFINITION OF TERMS

Ada Compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report.
Ada Implementation	An Ada compiler with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.

Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
LRM	The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>."
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90].
Validation	The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.
Withdrawn test	A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 12 October 1990.

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

2.2 INAPPLICABLE TESTS

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

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

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

The following 21 tests check for the predefined type LONG_INTEGER:

C35404C	C45231C	C45304C	C45411C	C45412C
C45502C	C45503C	C45504C	C45504F	C45611C
C45612C	C45613C	C45614C	C45631C	C45632C
B52004D	C55B07A	B55B09C	B86001W	C86006C
CD7101F				

C35702A, C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

C41401A checks that CONSTRAINT_ERROR is raised upon the evaluation of various attribute prefixes; this implementation derives the attribute values from the subtype of the prefix at compilation time, and thus does not evaluate the prefix or raise the exception. (See Section 2.3.)

C45531M..P (4 tests) and C45532M..P (4 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater.

C45624A checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 5. For this implementation, MACHINE_OVERFLOWS is TRUE.

C45624B checks that the proper exception is raised if MACHINE_OVERFLOWS is FALSE for floating point types with digits 6. For this implementation, MACHINE_OVERFLOWS is TRUE.

B86001Y checks for a predefined fixed-point type other than DURATION.

C86001F recompiles package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete. For this implementation, the package TEXT_IO is dependent upon package SYSTEM.

C96005B checks for values of type DURATION'BASE that are outside the range of DURATION. There are no such values for this implementation.

CD1009C uses a representation clause specifying a non-default size for a floating-point type.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use representation clauses specifying non-default sizes for access types.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions.

The tests listed in the following table are not applicable because the given file operations are not supported for the given combination of mode and file access method.

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

CE2107C..D (2 tests), CE2107H, and CE2107L apply function NAME to temporary sequential, direct, and text files in an attempt to associate multiple internal files with the same external file; USE_ERROR is raised because temporary files have no name.

CE2108B, CE2108D, and CE3112B use the names of temporary sequential, direct, and text files that were created in other tests in order to check that the temporary files are not accessible after the completion of those tests; for this implementation, temporary files have no name.

CE2203A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for SEQUENTIAL_IO. This implementation does not restrict file capacity.

EE2401D checks whether DIRECT_IO can be instantiated for an element type that is an unconstrained array type; this implementation raises USE_ERROR on the attempt to create a file, because the maximum potential element size exceeds the implementation limit of 2**31 - 1 bits.

CE2403A checks that WRITE raises USE_ERROR if the capacity of the external file is exceeded for DIRECT_IO. This implementation does not restrict file capacity.

CE3111B and CE3115A associate multiple internal text files with the same external file and attempt to read from one file what was written to the other, which is assumed to be immediately available; this implementation buffers output. (See section 2.3.)

CE3202A expects that function NAME can be applied to the standard input and output files; in this implementation these files have no names, and USE_ERROR is raised. (See section 2.3.)

CE3304A checks that USE_ERROR is raised if a call to SET_LINE_LENGTH or SET_PAGE_LENGTH specifies a value that is inappropriate for the external file. This implementation does not have inappropriate values for either line length or page length.

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

2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 12 tests.

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

B29001A BC2001D BC2001E BC3204B BC3205B BC3205D

C34007P and C34007S were graded passed by Evaluation Modification as directed by the AVO. These tests include a check that the evaluation of the selector "all" raises CONSTRAINT_ERROR when the value of the object is null. This implementation determines the result of the equality tests at lines 207 and 223, respectively, based on the subtype of the object; thus, the selector is not evaluated and no exception is raised, as allowed by LRM 11.6(7). The tests were graded passed given that their only output from Report.Failed was the message "NO EXCEPTION FOR NULL.ALL - 2".

C41401A was graded inapplicable by Evaluation Modification as directed by the AVO. This test checks that the evaluation of attribute prefixes that denote variables of an access type raises CONSTRAINT_ERROR when the value

of the variable is null and the attribute is appropriate for an array or task type. This implementation derives the array attribute values from the subtype; thus, the prefix is not evaluated and no exception is raised as allowed by LRM 11.6(7), for the checks at lines 77, 87, 106, 121, 131, 141, 152, 165, and 175.

CE3111B and CE3115A were graded inapplicable by Evaluation Modification as directed by the AVO. The tests assume that output from one internal file is unbuffered and may be immediately read by another file that shares the same external file. This implementation raises END_ERROR on the attempts to read at lines 87 and 101, respectively.

CE3202A was graded inapplicable by Evaluation Modification as directed by the AVO. This test applies function NAME to the standard input file, which in this implementation has no name; USE_ERROR is raised but not handled, so the test is aborted. The AVO ruled that this behavior is acceptable pending any resolution of the issue by the ARG.

CHAPTER 3

PROCESSING INFORMATION

3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For a point of contact for technical information about this Ada implementation system, see:

Michael Devlin
106 Apple Street
Tinton Falls NJ 07724
(201) 758-7531

For a point of contact for sales information about this Ada implementation system, see:

Michael Devlin
106 Apple Street
Tinton Falls NJ 07724
(201) 758-7531

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

3.2 SUMMARY OF TEST RESULTS

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

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

PROCESSING INFORMATION

Total Number of Applicable Tests	3799
Total Number of Withdrawn Tests	81
Processed Inapplicable Tests	89
Non-Processed I/O Tests	0
Non-Processed Floating-Point Precision Tests	201
Total Number of Inapplicable Tests	290
Total Number of Tests for ACVC 1.11	4170

All I/O tests of the test suite were processed because this implementation supports a file system. The above number of floating-point tests were not processed because they used floating-point precision exceeding that supported by the implementation.

3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 290 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were transferred directly to the host machine.

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

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

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

Option	Effect
INFORM	All the information messages (if applicable) appear in the listing file.
INLINE	All the pragma Inline requests are honored (when applicable).
LIST	A listing file is produced.
OPTIMIZE	The compiler performs some optimizations to improve run-time efficiency of the program.
PAGE_SIZE	Specifies that there will be X lines per page in the listing file. (Set at 60).
SEGMENTED	The object code will contain both PURE (read only) and IMPURE (read and write) segments.
STACK_CHECK	The compiler generates extra code to check if there is enough stack space.
SUMMARY	The compiler generates a summary file when adacomp is used with $file_list_name.
WARN	All the warning messages appear in the listing file.

The listings were printed on a remote system via a remote shell call. Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

APPENDIX A

MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

Macro Parameter	Macro Value
$MAX_IN_LEN	255
$BIG_ID1	(1..V-1 => 'A', V => '1')
$BIG_ID2	(1..V-1 => 'A', V => '2')
$BIG_ID3	(1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A')
$BIG_ID4	(1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A')
$BIG_INT_LIT	(1..V-3 => '0') & "298"
$BIG_REAL_LIT	(1..V-5 => '0') & "690.0"
$BIG_STRING1	'"' & (1..V/2 => 'A') & '"'
$BIG_STRING2	'"' & (1..V-1-V/2 => 'A') & '1' & '"'
$BLANKS	(1..V-20 => ' ')
$MAX_LEN_INT_BASED_LITERAL	"2:" & (1..V-5 => '0') & "11:"
$MAX_LEN_REAL_BASED_LITERAL	"16:" & (1..V-7 => '0') & "F.E:"

```
$MAX_STRING_LITERAL    '"' & (1..V-2 => 'A') & '"'
```

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

Macro Parameter	Macro Value
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	(2**31)-1
$DEFAULT_MEM_SIZE	(2**24)
$DEFAULT_STOR_UNIT	8
$DEFAULT_SYS_NAME	CCUR_3200
$DELTA_DOC	2#1.0#E-31
$ENTRY_ADDRESS	16#40#
$ENTRY_ADDRESS1	16#80#
$ENTRY_ADDRESS2	16#100#
$FIELD_LAST	255
$FILE_TERMINATOR	' '
$FIXED_NAME	NO_SUCH_FIXED_TYPE
$FLOAT_NAME	NO_SUCH_TYPE
$FORM_STRING	"RS=>80"
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	100000.0
$GREATER_THAN_DURATION_BASE_LAST	131073.0
$GREATER_THAN_FLOAT_BASE_LAST	1.8E+63
$GREATER_THAN_FLOAT_SAFE_LARGE	16#0.FFFFFA#E63

```
$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
                        0.0

$HIGH_PRIORITY          255

$ILLEGAL_EXTERNAL_FILE_NAME1
                        \NODIRECTORY\FILENAME

$ILLEGAL_EXTERNAL_FILE_NAME2
                        THIS_FILE_NAME_IS_TOO_LONG_FOR_MY_SYSTEM

$INAPPROPRIATE_LINE_LENGTH
                        -1

$INAPPROPRIATE_PAGE_LENGTH
                        -1

$INCLUDE_PRAGMA1        PRAGMA INCLUDE ("A28006D1.TST")

$INCLUDE_PRAGMA2        PRAGMA INCLUDE ("B28006D1.TST")

$INTEGER_FIRST          -2147483648

$INTEGER_LAST           2147483647

$INTEGER_LAST_PLUS_1    2147483648

$INTERFACE_LANGUAGE     FORTRAN

$LESS_THAN_DURATION     -75000.0

$LESS_THAN_DURATION_BASE_FIRST
                        -131073.0

$LINE_TERMINATOR        ' '

$LOW_PRIORITY           0

$MACHINE_CODE_STATEMENT
                        NULL;

$MACHINE_CODE_TYPE      NO_SUCH_TYPE

$MANTISSA_DOC           31

$MAX_DIGITS             15

$MAX_INT                2147483647

$MAX_INT_PLUS_1         2147483648

$MIN_INT                -2147483648
```

MACRO PARAMETERS

$NAME	TINY_INTEGER
$NAME_LIST	CCUR_3200
$NAME_SPECIFICATION1	ACVC:X2120A./P
$NAME_SPECIFICATION2	ACVC:X2120B./P
$NAME_SPECIFICATION3	ACVC:X3119A./P
$NEG_BASED_INT	16#F000000E#
$NEW_MEM_SIZE	(2**24)
$NEW_STOR_UNIT	8
$NEW_SYS_NAME	CCUR_3200
$PAGE_TERMINATOR	' '
$RECORD_DEFINITION	NEW INTEGER
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	1024
$TICK	0.01
$VARIABLE_ADDRESS	16#0020#
$VARIABLE_ADDRESS1	16#0024#
$VARIABLE_ADDRESS2	16#0028#
$YOUR_PRAGMA	INLINE

APPENDIX B

COMPILATION SYSTEM OPTIONS

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

Option	Effect
ABORT	If set, the compiler stops compiling when an error is encountered in a unit and will not compile any subsequent units in the source file.
ALIST	If set, a symbolic assembly listing will be appended to the listing file.
INFORM	If set, all the information messages (if applicable) will appear in the listing file.
INLINE	If set, all the pragma Inline requests will be honored (when applicable).
LIST	If set, a listing file will be produced.
OPTIMIZED	If set, the compiler performs some optimizations to improve run-time efficiency of the program.
PAGE_SIZE	The default specifies that there will be X lines per page in the listing file.
SEGMENTED	If set, the object code will contain both PURE (read only) and IMPURE (read and write) segments.

COMPILATION SYSTEM OPTIONS

Option	Effect
STACK_CHECK	If set, the compiler generates extra code to check if there is enough stack space.
SUMMARY	If set, the compiler generates a summary file when adacomp is used with a $file_list_name.
SUPPRESS_ALL	If set, the compiler will not generate any run-time checking code for all types and objects in the compilation: Access_check, Discriminant_check, Index_check, Length_check, Overflow_check, and Range_check.
SUPPRESS_OVERFLOW	If set, the Overflow checking code will not be generated by the compiler.
WARN	If set, all the warning messages will appear in the listing file.

APPENDIX C

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is
    ...

    type INTEGER is range -2147483648 .. 2147483647;

    type TINY_INTEGER is range -128 .. 127;

    type SHORT_INTEGER is range -32768 .. 32767;

    type FLOAT is digits 6 range -16#0.FFFF_F8#E63 .. 16#0.FFFF_F8#E63;

    type LONG_FLOAT is digits 15
         range -16#0.FFFF_FFFF_FFFF_FE#E63 .. 16#0.FFFF_FFFF_FFFF_FE#E63;

    type DURATION is delta 0.00006103515625
         range -131072.00 .. 131071.99993896484375;

    ...
end STANDARD;
```

APPENDIX F
IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 INTRODUCTION

The following sections provide all implementation-dependent characteristics of the C^{3}Ada compiler.

F.2 IMPLEMENTATION-DEPENDENT PRAGMAS

The following is the syntax representation of a pragma:

```
pragma identifier [(argument [,argument])];
```

Where:

identifier is the name of the pragma.

argument defines a parameter of the pragma. For example, the LIST pragma expects the arguments ON or OFF.

PRAGMA	IMPLEMENTED	COMMENTS
BIT_PACK	Yes	This pragma allows packing of composite objects to the bit level, thereby achieving greater data compaction.
BYTE_PACK	Yes	The elements of an array or record are packed down to a minimal number of bytes.
CONTROLLED	No	Not applicable because no automatic storage reclamation of unreferenced access objects is performed.
ELABORATE	Yes	Is handled as defined by the Ada language.
EXTERNAL_NAME	Yes	Defines the link-time name of a statically allocated object or subprogram.
INLINE	Yes	Is handled as defined by the Ada language, with the following restrictions: The subprogram to be expanded inline must not contain declarations of other subprograms, task bodies, generic units, or body stubs. If the subprogram is called recursively, only the outer call is expanded. The subprogram must be previously compiled, and if it is a generic instance, it must be previously completed.

PRAGMA	IMPLEMENTED	COMMENTS
INTERFACE	Yes	Is implemented for ASSEMBLER and FORTRAN.
LIST	Yes	Is handled as defined by the Ada language.
MEMORY_SIZE	No	You cannot specify the number of available storage units in the machine configuration, which is defined in package SYSTEM.
OPTIMIZE	No	You cannot specify either time or space as the primary optimization criterion. However, OPTIMIZE is accepted as a compile-time option.
PACK	Yes	The elements of an array or record are packed down to a minimal number of bits.
PAGE	Yes	Is handled as defined by the Ada language.
PARTIAL_IMAGE	Yes	This pragma informs the compiler that the named package may be used to build a partial image and causes the compiler to verify that the package meets all requirements for such use.
PRIORITY	Yes	Is implemented as defined by the Ada language.
REENTRANT_LIBRARY	Yes	This pragma informs the compiler that the unit may be used in a re-entrant library, sharing code but not data; the compiler checks that the unit meets all requirements for such use.
SHARED	Yes	Is implemented as defined by the Ada language.
STACK_CHECK	Yes	When specified with OFF, this pragma indicates to the compiler that there is enough space in the initial stack chunk for the activation record of the subprograms. Therefore, no code is generated to check for the need to provide additional space for the run-time stack of that subprogram.
STORAGE_UNIT	No	You cannot specify the number of bits per storage unit, which is defined in package SYSTEM.
SUPPRESS	No	Different types of checks cannot be switched on or off for specific objects; however, see SUPPRESS_ALL.
SUPPRESS_ALL	Yes	This pragma allows the compiler to omit the generation of code to check for errors that may raise CONSTRAINT_ERROR or PROGRAM_ERROR that may be raised due to an elaboration order problem.

PRAGMA	IMPLEMENTED	COMMENTS
SYSTEM_NAME	No	You cannot alter the target system name, which is defined in package SYSTEM.
VOLATILE	Yes	Specifies that every read and update of a variable causes a reference to the actual memory location of the variable. That is, a local copy of the variable is never made. This is similar to the pragma SHARED, except that any variable may be specified and that it does not cause synchronization of tasks. This pragma suppresses all optimizations on the specified variable.

F.2.1 Pragma INLINE Restrictions

Inline expansion of a subprogram call occurs if the following conditions are satisfied:

- If the subprogram body is not contained in the same compilation unit as the call, then the compilation unit containing the body must be successfully compiled before the unit containing the call.
- The subprogram body and any enclosed declare blocks do not contain
  - declarations of subprograms,
  - declarations of task types or single tasks,
  - body stubs, or
  - generic instantiations.

Note that if a call is expanded inline in one compilation unit and the subprogram body is in another compilation unit, then the former unit will be made obsolete if the latter unit is recompiled.

For every call of a subprogram for which pragma INLINE is given, a warning message is reported if the above set of conditions is not satisfied; the message indicates that inline expansion is not done for the particular call.

Inline expansion occurs when the expanded code contains a valid subprogram call. However, inline expansion is not carried out for a subprogram call if inline expansion for the same subprogram is already in progress, e.g. for a recursive call. A warning message is generated informing the user of such a situation.

Finally, when you specify pragma INLINE for procedures and/or functions in generic units, the compiler inlines calls to instances of them only if the bodies of the instances are already completed using the Completer.

F.3 LENGTH CLAUSES

A length clause specifies the amount of storage associated with a given type. The following is a list of the implementation-dependent attributes.

T'SIZE — must be 32 for a type derived from FLOAT and 64 for a type derived from LONG_FLOAT. For array and record types, only the size chosen by the compiler may be specified.

T'STORAGE_SIZE — is fully supported for collection size specification.

T'STORAGE_SIZE is not supported for task activation. Task memory is limited by the work space for the program.

Size representation only applies to types – not to subtypes. In the following example, the size of T is 32, but the size of T1 is not necessarily 32.

```
type T is range  0..100;
subtype T1 is T range 0..10;
for T'SIZE use 32;
```

In the following example, the size of the subtype is the same as the size of the type (size of the type is applied to the subtype).

```
type T is range  0..100;
for T'SIZE use 32;
subtype T2 is T range 0..10;
```

F.4 REPRESENTATION ATTRIBUTES

The Representation attributes listed below are as described in the *Reference Manual for the Ada® Programming Language*, Section 13.7.2.

X'ADDRESS	is not supported for labels and packages.
X'SIZE	is handled as defined by the Ada Language.
R.C'POSITION	is handled as defined by the Ada Language.
R.C'FIRST_BIT	is handled as defined by the Ada Language.
R.C'LAST_BIT	is handled as defined by the Ada Language.
T'STORAGE_SIZE	for access types, returns the current amount of storage reserved for the type. If a T'STORAGE_SIZE representation clause is specified, then the amount specified is returned; otherwise, the current amount allocated is returned.
T'STORAGE_SIZE	for task types or objects is not implemented. It returns 0.

F.4.1 Representation Attributes of Real Types

P'DIGITS	yields the number of decimal digits for the subtype P. This value is 6 for type FLOAT and 15 for type LONG_FLOAT.
P'MANTISSA	yields the number of binary digits in the mantissa of P. The value is 21 for type FLOAT and 51 for type LONG_FLOAT.

DIGITS	MANTISSA	DIGITS	MANTISSA	DIGITS	MANTISSA
1	5	6	21	11	38
2	8	7	25	12	41
3	11	8	28	13	45
4	15	9	31	14	48
5	18	10	35	15	51

P'EMAX yields the largest exponent value of model numbers for the subtype P. The value is 84 for type FLOAT and 204 for type LONG_FLOAT.

DIGITS	EMAX	DIGITS	EMAX	DIGITS	EMAX
1	20	6	84	11	152
2	32	7	100	12	164
3	44	8	112	13	180
4	60	9	124	14	192
5	72	10	140	15	204

P'EPSILON yields the absolute value of the difference between the model number 1.0 and the next model number above for the subtype P. The value is 16#0.0000_1# for type FLOAT and 16#0.0000_0000_0000_4# for type LONG_FLOAT.

DIGITS	EPSILON	DIGITS	EPSILON	DIGITS	EPSILON
1	16#0.1#E00	6	16#0.1#E-4	11	16#0.8#E-9
2	16#0.2#E-1	7	16#0.1#E-5	12	16#0.1#E-9
3	16#0.4#E-2	8	16#0.2#E-6	13	16#0.1#E-10
4	16#0.4#E-3	9	16#0.4#E-7	14	16#0.2#E-11
5	16#0.8#E-4	10	16#0.4#E-8	15	16#0.4#E-12

P'SMALL yields the smallest positive model number of the subtype P. The value is 16#0.8#E-21 for type FLOAT and 16#0.8#E-51 for type LONG_FLOAT.

VALUES	SMALL	VALUES	SMALL	VALUES	SMALL
1	16#0.8#E-5	6	16#0.8#E-21	11	16#0.8#E-38
2	16#0.8#E-8	7	16#0.8#E-25	12	16#0.8#E-41
3	16#0.8#E-11	8	16#0.8#E-28	13	16#0.8#E-45
4	16#0.8#E-15	9	16#0.8#E-31	14	16#0.8#E-48
5	16#0.8#E-18	10	16#0.8#E-35	15	16#0.8#E-51

P'LARGE yields the largest positive model number of the subtype P. The value is 16#0.FFFF_F8#E21 for type FLOAT and 16#0.FFFF_FFFF_FFFF_E#E51 for type LONG_FLOAT.

VALUES	LARGE
1	16#0.F8#E5
2	16#0.FF#E8
3	16#0.FFE#E11
4	16#0.FFFE#E15
5	16#0.FFFF_C#E18
6	16#0.FFFF_F8#E21
7	16#0.FFFF_FF8#E25
8	16#0.FFFF_FFF#E28
9	16#0.FFFF_FFFE#E31
10	16#0.FFFF_FFFF_E#E35
11	16#0.FFFF_FFFF_FC#E38
12	16#0.FFFF_FFFF_FF8#E41
13	16#0.FFFF_FFFF_FFF8#E45
14	16#0.FFFF_FFFF_FFFF#E48
15	16#0.FFFF_FFFF_FFFF_E#E51

P'SAFE_EMAX — yields the largest exponent value of safe numbers of type P. The value is 252 for types FLOAT and LONG_FLOAT.

P'SAFE_SMALL — yields the smallest positive safe number of type P. The value is 16#0.8#E-63 for types FLOAT and LONG_FLOAT.

P'SAFE_LARGE — yields the largest positive safe number of the type P. The value is 16#0.FFFF_F8#E63 for type FLOAT and 16#0.FFFF_FFFF_FFFF_FE#E63 for type LONG_FLOAT.

P'MACHINE_ROUNDS is true.

P'MACHINE_OVERFLOWS is true.

P'MACHINE_RADIX is 16.

P'MACHINE_MANTISSA is six for types derived from FLOAT; else 14.

P'MACHINE_EMAX is 63.

P'MACHINE_EMIN is -64.

F.4.2 Representation Attributes of Fixed Point Types

For any fixed point type T, the representation attributes are:

T'MACHINE_ROUNDS	true
T'MACHINE_OVERFLOWS	true
T'MANTISSA	1..31
T'SIZE	2..32

F.4.3 Enumeration Representation Clauses

The maximum number of elements in an enumeration type is limited by the maximum size of the enumeration image table which cannot be greater than 65535 bytes. The enumeration table size is determined by the following function:

```
generic
   type ENUMERATION_TYPE is (<>);
function ENUMERATION_TABLE_SIZE return NATURAL is
   RESULT : NATURAL :=0;
begin
   for I in ENUMERATION_TYPE 'FIRST..ENUMERATION_TYPE' LAST loop
       RESULT :=RESULT + 2 + I'WIDTH;
   end loop;
   return RESULT;
end ENUMERATION_TABLE_SIZE;
```

F.4.4 Record Representation Clauses

The *Reference Manual for the Ada® Programming Language* states that an implementation may generate names that denote implementation-dependent components. This is not present in this release of the C^{3}Ada compiler. Implementation-dependent offset components are created for record components whose size is dynamic or dependent on discriminants. These offset components have no names.

Record components of a private type cannot be included in a record representation specification.

Record clause alignment can only be 1, 2, or 4.

Component representations for access types must allow for at least 24 bits.

Component representations for scalar types other than for types derived from LONG_FLOAT must not specify more than 32 bits.

F.4.5 Type DURATION

DURATION'SMALL equals 61.035μs or 2^{-14} seconds. This number is the smallest power of two which can still represent the number of seconds in a day in a fullword fixed point number.

System.tick equals 10ms. The actual computer clock-tick is 1.0/120.0 seconds (or about 8.33333ms) in 60Hz areas and 1.0/100.0 seconds (or 10ms) in 50Hz areas. System.tick represents the greater of the actual clock-tick from both areas.

DURATION'SMALL is significantly smaller than the actual computer clock-tick. Therefore, the least amount of delay possible is limited by the actual clock-tick. The delay of DURATION'SMALL follows this formula:

<actual-clock-tick> ± <actual-clock-tick> + 1.3μs

The 1.3μs represents the overhead or the minimum delay possible on a 3280MPS. For 60Hz areas, the range of delay is approximately from 1.3μs to 17.97μs. For 50Hz areas, the range of delay is approximately from 1.3μs to 21.3μs. However, on the average, the delay is slightly greater than the actual clock-tick.

In general, the formula for finding the range of a delay value, *x*, is:

nearest_multiple(*x*,<actual-clock-tick>) ± <actual-clock-tick> + 1.3μs

where *nearest_multiple* rounds *x* up to the nearest multiple of the actual clock-tick.

TABLE F-1. TYPE DURATION

ATTRIBUTE	VALUE	APPROXIMATE VALUE
DURATION'DELTA	2#1.0#E-14	≃ 61μs
DURATION'SMALL	2#1.0#E-14	≃ 61μs
DURATION'FIRST	-131072.00	≃ 36 hrs
DURATION'LAST	131071.99993896484375	≃ 36 hrs
DURATION'SIZE	32	

F.5 ADDRESS CLAUSES

Address clauses are implemented for objects. No storage is allocated for objects with address clauses by the compiler. The user must guarantee the storage for these by some other means (e.g., through the use of the absolute instruction found in the *Common Assembly Language/32 (CAL/32) Reference Manual*). The exception PROGRAM_ERROR is raised upon reference to the object if the specified address is not in the program's address space or is not properly aligned.

Address clauses are supported for task entries. If an address clause is specified for subprograms, packages, task objects, or task units, the compiler will generate a restriction error.

An address clause applied to a task entry enables an operating system trap to initiate an entry call to that entry. The address specified is an address clause for a task entry can be done via a function call INTERRUPT_ADDRESS with QUEUE_EVENT as its argument. Function

INTERRUPT_ADDRESS is specified in package OS_32_REAL_TIME_SERVICES.

Initialization of an object that has an address clause specified is not supported. Objects with address clauses may also be used to map objects into task common (TCOM) areas. See Chapter 4 for more information regarding task common.

F.6 THE PACKAGE SYSTEM

The package SYSTEM, provided with the C^{3}Ada system, permits access to machine-dependent features. The specification of the package SYSTEM declares constant values dependent on the Series 3200 Processors. The following is a listing of the visible section of the package SYSTEM specification.

```
package SYSTEM is

   type DESIGNATED_BY_ADDRESS is private;
   type ADDRESS is access DESIGNATED_BY_ADDRESS;
   ADDRESS_ZERO : constant ADDRESS := NULL;
   ADDRESS_NULL : constant ADDRESS := NULL;

   function "+" (LEFT : ADDRESS; RIGHT : INTEGER) return ADDRESS;
   function "+" (LEFT : INTEGER; RIGHT : ADDRESS) return ADDRESS;
   function "-" (LEFT : ADDRESS; RIGHT : INTEGER) return ADDRESS;
   function "-" (LEFT : ADDRESS; RIGHT : ADDRESS) return INTEGER;

   type NAME is (CCUR_3200);
   SYSTEM_NAME  : constant NAME := CCUR_3200;
   STORAGE_UNIT : constant := 8;
   MEMORY_SIZE  : constant := 2 ** 24;
   MIN_INT      : constant := - 2_147_483_648;
   MAX_INT      : constant := 2_147_483_647;
   MAX_DIGITS   : constant := 15;
   MAX_MANTISSA : constant := 31;
   FINE_DELTA   : constant := 2.0 ** (-31);
   TICK         : constant := 0.01;

   type UNSIGNED_SHORT_INTEGER is range 0 ..  65_535;
   type UNSIGNED_TINY_INTEGER is range 0 .. 255;
   for UNSIGNED_SHORT_INTEGER'size use 16;
   for UNSIGNED_TINY_INTEGER'size use 8;
   subtype BYTE is UNSIGNED_TINY_INTEGER;
   subtype ADDRESS_RANGE is INTEGER range 0 .. 2**24-1;
   subtype PRIORITY is INTEGER range 0 .. 255;
   type SEMAPHORE_MODE is (NOT_EXECUTED, EXECUTED);
   type SEMAPHORE is private;

   --These functions are for compatibility with previous versions of
   --C3Ada, and should be replaced with calls to unchecked_conversion.
   function COPY_DOUBLEWORD (FROM: LONG_FLOAT) return LONG_FLOAT;
     pragma INLINE (COPY_DOUBLEWORD);
   function COPY_FULLWORD (FROM: INTEGER) return ADDRESS;
     pragma INLINE (COPY_FULLWORD);
   function COPY_FULLWORD (FROM: ADDRESS) return INTEGER;
     pragma INLINE (COPY_FULLWORD);
   function COPY_HALFWORD (FROM: SHORT_INTEGER) return SHORT_INTEGER;
     pragma INLINE (COPY_HALFWORD);
   function COPY_BYTE (FROM : TINY_INTEGER) return TINY_INTEGER;
     pragma INLINE (COPY_BYTE);
   function MEMORY_USED return NATURAL;
     pragma INTERFACE (ASSEMBLER, MEMORY_USED);
   function HEAP_USED return NATURAL;
```

```
   pragma INTERFACE (ASSEMBLER, HEAP_USED);

--Address conversion routines
function INTEGER_TO_ADDRESS (ADDR : ADDRESS_RANGE) return ADDRESS
 renames COPY_FULLWORD;
function ADDRESS_TO_INTEGER (ADDR : ADDRESS) return ADDRESS_RANGE
 renames COPY_FULLWORD;

procedure EXECUTE_OR_WAIT (S: in out SEMAPHORE; S_MODE: out SEMAPHORE);
procedure COMPLETED_EXECUTION (S: in out SEMAPHORE);
procedure RESET_SEMAPHORE (S: in out SEMAPHORE);

type EXCEPTION_ID IS new INTEGER;
NO_EXCEPTION_ID : constant EXCEPTION_ID := 0;

type EXIT_CODE is new INTEGER range 0..235;
ERROR : constant EXIT_CODE := 254;
SUCCESS : constant EXIT_CODE := 0;

function LAST_EXCEPTION_ID return EXCEPTION_ID;


CONSTRAINT_ERROR_ID : constant EXCEPTION_ID
   := implementation defined

NUMERIC_ERROR_ID    : constant EXCEPTION_ID
   := implementation defined

PROGRAM_ERROR_ID    : constant EXCEPTION_ID
   := implementation defined

STORAGE_ERROR_ID    : constant EXCEPTION_ID
   := implementation defined

TASKING_ERROR_ID    : constant EXCEPTION_ID
   := implementation defined

STATUS_ERROR_ID     : constant EXCEPTION_ID
   := implementation defined

MODE_ERROR_ID       : constant EXCEPTION_ID
   := implementation defined

NAME_ERROR_ID       : constant EXCEPTION_ID
   := implementation defined

USE_ERROR_ID        : constant EXCEPTION_ID
   := implementation defined

DEVICE_ERROR_ID     : constant EXCEPTION_ID
   := implementation defined

END_ERROR_ID        : constant EXCEPTION_ID
   := implementation defined

DATA_ERROR_ID       : constant EXCEPTION_ID
   := implementation defined

LAYOUT_ERROR_ID     : constant EXCEPTION_ID
   := implementation defined
```

```
   TIME_ERROR_ID        : constant EXCEPTION_ID
      := implementation defined

   function MEMORY_USED return NATURAL;
   function HEAP_USED return NATURAL;

private
   -- Implementation defined

   type SEMAPHORE is
      record
         SEMA_OBJ : INTEGER := 0;
      end record;
   pragma INTERFACE (ASSEMBLER, EXECUTE_OR_WAIT);
   pragma INTERFACE (ASSEMBLER, COMPLETED_EXECUTION);
   pragma INTERFACE (ASSEMBLER, RESET_SEMAPHORE);

end SYSTEM;
```

F.7 INTERFACE TO OTHER LANGUAGES

Pragma INTERFACE is implemented for two languages, Assembly and FORTRAN. The pragma can take one of three forms:

- For any Assembly language procedure or function:

```
pragma INTERFACE (ASSEMBLER, ROUTINE_NAME);
```

- For FORTRAN functions with only `in` parameters or procedures:

```
pragma INTERFACE (FORTRAN, ROUTINE_NAME);
```

- For FORTRAN functions that have `in out` or `out` parameters:

```
pragma INTERFACE (FORTRAN, ROUTINE_NAME, IS_FUNCTION);
```

In the C^{3}Ada system, functions cannot have `in out` or `out` parameters so the Ada specification for the function is written as a procedure with the first argument being the function return result. Then, the parameter IS_FUNCTION is specified to inform the compiler that it is, in reality, a FORTRAN function. Interface routine_names are truncated to an 8 character maximum length.

F.8 INPUT/OUTPUT (I/O) PACKAGES

The following two system-dependent parameters are used for the control of external files:

- NAME parameter
- FORM parameter

The NAME parameter must be an OS/32 filename string. Figure F-1 illustrates the four fields in an OS/32 file descriptor.

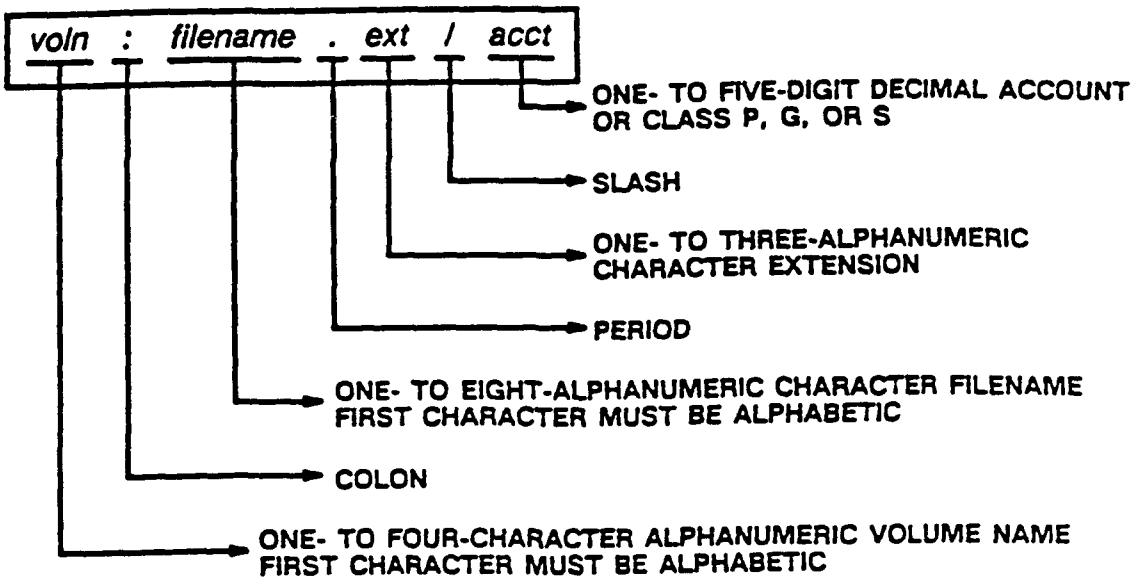

Figure F-1. OS/32 File Descriptor

The implementation-dependent values used for keywords in the FORM parameter are discussed below. The FORM parameter is a string that contains further system-dependent characteristics and attributes of an external file. The FORM parameter conveys to the file system information on the intended use of the associated external file. This parameter is used as one of the specifications for the CREATE procedure and the OPEN procedure. It specifies a number of system-dependent characteristics such as lu, file format, etc. It is returned by the FORM function.

The syntax of the FORM string, in our implementation, uses Ada syntax conventions and is as follows:

```
form_param  ::= [form_spec [, form_spec)]
form_spec   ::= lu_spec | fo_spec |
                rs_spec | dbf_spec |
                ibf_spec | al_spec |
                pr_spec | keys_spec |
                pad_spec | dc_spec |
                da_spec | ds_spec |
                ps_spec | ch_spec
lu_spec     ::= LU => lu
fo_spec     ::= FILE_ORGANIZATION => fo
rs_spec     ::= RECORD_SIZE => rs
dbf_spec    ::= DATA_BLOCKING_FACTOR => dbf
ibf_spec    ::= INDEX_BLOCKING_FACTOR => ibf
al_spec     ::= ALLOCATION => al
pr_spec     ::= PRIVILEGE => pr
keys_spec   ::= KEYS => keys
pad_spec    ::= PAD => pad
dc_spec     ::= DEVICE_CODE => dc
da_spec     ::= DEVICE_ATTRIBUTE => da
ds_spec     ::= DEVICE_STATUS => ds
ps_spec     ::= PROMPTING_STRING => ps
ch_spec     ::= CHARACTER_IO
```

The exception USE_ERROR is raised if a given FORM parameter string does not have the correct syntax or if certain conditions concerning the OPEN or CREATE statements are not fulfilled. Keywords that are listed above in upper-case letters are also recognized by the compiler in lower-case.

lu is an integer in the range 0..254 specifying the lu number.

fo specifies legal OS/32 file formats (file organization). They are:

```
INDEX | IN
| CONTIGUOUS | CO
| NON_BUFFERED | NB
| EXTENDABLE_CONTIGUOUS | EC
| LONG_RECORD | LR
| ITAM
| DEVICE
```

rs is an integer in the range 1..65535 specifying the physical record size.

- For INDEX, ITAM, and NON_BUFFERED files, this specifies the physical record size.
- The physical record size for CONTIGUOUS and EXTENDABLE_CONTIGUOUS files is determined by rounding the element size up to the nearest 256-byte boundary. For such files, *rs* is ignored.
- The physical record size for LONG_RECORD files is specified by the data blocking factor multiplied by 256 and *rs* is ignored.
- For a DEVICE, the physical record size always equals the element size and *rs* is ignored.

dbf specifies the data blocking factor. An integer in the range 0..255 (maximum set up at OS/32 system generation (sysgen) time) that specifies the number of contiguous disk sectors (256 bytes) in a data block. It applies only to INDEX, NON_BUFFERED, EXTENDABLE_CONTIGUOUS, and LONG_RECORD files. For other file organizations (see *fo* above), it is ignored. The default is 0, which causes the data blocking factor to be set to the current OS/32 default.

ibf specifies the index blocking factor. An integer in the range 0..255 (maximum set up at OS/32 sysgen time) specifying the number of contiguous disk sectors (256 bytes) in an index block of an INDEX, NON_BUFFERED, EXTENDABLE_CONTIGUOUS, or LONG_RECORD file. The default is 0, which causes the index blocking factor to be set the current OS/32 default. For other file organizations (see *fo*), it is ignored.

al specifies an integer in the range 1..2_147_483_647. For CONTIGUOUS files, it specifies the number of 256-byte sectors to be allocated. For ITAM files, it specifies the physical block size in bytes associated with the buffered terminal. For other file organizations (see *fo*), it is ignored.

pr specifies OS/32 access privileges, i.e., SRO, ERO, SWO, EWO, SRW, SREW, ERSW, and ERW.

keys is a decimal or hexadecimal integer specifying the OS/32 READ/WRITE keys, in range 16#0000# .. 16#FFFF# (0..65535). The left two hexadecimal digits signify the write protection key and the right two hexadecimal digits signify the read protection key.

pad specifies the padding character used for READ and WRITE operations; the pad character is either NONE, BLANK, or NUL. The default is NONE.

If the pad characters are NONE, the records are not padded. If NUL, records are padded with ASCII.NULL. If BLANK, records are padded with blank and OS/32 ASCII I/O operations are used.

dc is an integer in the range 0..255 specifying the OS/32 device code of the external file. See the *System Generation/32 (Sysgen/32) Reference Manual* for a list of all devices and their respective codes. A device code specifier is ignored for OPEN or CREATE requests. See function FORM in Section 12.5.9.

da is an integer in the range 0..65535 specifying the OS/32 device attributes of the external file. See the *OS/32 Supervisor Call (SVC) Reference Manual* (the table on description and mask values of the device attributes field). A device attribute specifier is ignored for OPEN or CREATE requests. See function FORM in Section 12.5.9.

ds is an integer in the range 0..65535 specifying the status of the external file. A status of 0 means that the access to the file terminated with no errors; otherwise, a device error occurred. For errors occurring during READ and WRITE operations, the status values and their meanings are found in Chapter 2 of the *OS/32 Supervisor Call (SVC) Reference Manual* (the tables on device-independent and device-dependent status codes). A device status specifier is ignored for OPEN or CREATE requests. See function FORM in Section 12.5.9.

ps is a string of up to 80 characters, enclosed in quotation marks (" "), that specifies the prompting string. Embedded quotation marks must be doubled. (See Section 12.6.1 on CREATE and OPEN for TEXT_IO for more information.) For example:

```
form => "prompting_string =>""Process 1"""
```

F.8.1 Text Input/Output (I/O)

There are two implementation-dependent types for TEXT_IO: COUNT and FIELD. Their declarations implemented for the C^{3}Ada compiler are as follows:

```
type COUNT is range 0 ..INTEGER'LAST;
subtype FIELD is INTEGER range 0 ..512;
```

F.8.1.1 End of File Markers

In this implementation, the line terminator, page terminator and file terminator are not represented as a sequence of ASCII control characters.

F.8.2 Restrictions on ELEMENT_TYPE

The following are the restrictions concerning ELEMENT_TYPE:

- I/O of access types is undefined, although allowable; i.e., the fundamental association between the access variable and its accessed type is ignored.
- The maximum size of a variant data type is always used.
- If the size of the element type is exceeded by the physical record length, then during a READ operation the extra data on the physical record is lost. The exception DATA_ERROR is not raised.

- If the size of the element type exceeds the physical record length during a WRITE operation, the extra data in the element is not transferred to the external file and DATA_ERROR is not raised.
- SEQUENTIAL_IO and DIRECT_IO can be instantiated with unconstrained types. However, this causes the file to have a record length that is the maximum size of the unconstrained type. If this record length is larger than what is supported for the type of file involved, the exception USE_ERROR is raised.
- I/O operations on composite types containing dynamic array components will not transfer these components because they are not physically contained within the record itself.

F.8.3 TEXT Input/Output (I/O) on a Terminal

A line terminator is detected when an ASCII.CR is input or output or the operating system detects a full buffer. No spanned records with ASCII.NUL are output.

A line terminator followed by a page terminator may be represented as:

```
ASCII.CR
ASCII.FF ASCII.CR
```

if they are issued separately by the user, e.g., NEW_LINE followed by a NEW_PAGE. The same reasoning applies for a line terminator followed by a page terminator, which is then followed by a file terminator.

All text I/O operations are buffered unless form CHARACTER_IO is specified. This means that physical I/O operations are performed on a line by line basis, as opposed to a character by character basis. For example:

```
put ("Enter Data");
get_line (data, len);
```

will not output the string "Enter Data" until the next PUT_LINE or NEW_LINE operation is performed.

F.9 MACHINE CODE INSERTIONS

Code statements are not supported.

F.10 UNCHECKED PROGRAMMING

Unchecked programming gives the programmer the ability to circumvent some of the strong typing and elaboration rules of the Ada language. As such, it is the programmer's responsibility to ensure that the guidelines provided in the following sections are followed.

F.10.1 Unchecked Storage Deallocation

The unchecked storage deallocation generic procedure explicitly deallocates the space for a dynamically acquired object.

Restrictions:

This procedure frees storage only if:

- the object being deallocated was the last one allocated of all objects in a given declarative part; or

- all objects in a single chunk of the collection belonging to all access types declared in the same declarative part are deallocated.

F.10.2 *Unchecked Type Conversions*

The unchecked type conversion generic function permits the user to convert, without type checking, from one type to another. It is the user's responsibility to guarantee that such a conversion preserves the properties of the target type.

Restrictions:

The object used as the parameter in the function may not have components which contain dynamic or unconstrained array types.

If the TARGET'SIZE is greater than the SOURCE'SIZE, the resulting conversion is unpredictable. If the TARGET'SIZE is less than the SOURCE'SIZE, the result is that the left-most bits of the source are placed in the target.

Since UNCHECKED_CONVERSION is implemented as an arbitrary block move, no alignment constraints are necessary on the source or the target operands.

F.11 IMPLEMENTATION-DEPENDENT RESTRICTIONS

- The main procedure must be parameterless.
- The source line length must be less than or equal to 255 characters.
- Due to the source line length, the largest identifier is 255 characters.
- No more than 65534 lines per compilation unit.
- The maximum number of library units is 16380.
- The maximum number of bits in an object is 2^{31} -1.
- The maximum static nesting level is 63.
- The maximum number of directly imported units of a single compilation unit must not exceed 1024.
- Recompilation of SYSTEM or CALENDAR specification is prohibited.
- ENTRY'ADDRESS, PACKAGE'ADDRESS, and LABEL'ADDRESS are not supported.
- The maximum number of nested SEPARATES is 511.
- The maximum length of a filename is 80 characters.
- The maximum length of a program library name is 64 characters.
- The maximum length of a listing line is 132 characters.
- The maximum number of errors handled is 1000.
- The maximum subprogram nesting level is 64.
- The maximum number of calls to pragma ELABORATE per compilation unit is 1024.
- The total size for text of unique symbols (including imported symbols) per compilation is 300000.
- The maximum parser stack depth is 10000.
- The maximum depth of packages is 511.
- The static aggregate nesting limit is 256.

F.12 UNCONSTRAINED RECORD REPRESENTATIONS

Objects of an unconstrained record type with array components based on the discriminant are allocated using the discriminant value supplied in the object declaration. If the size of

an unconstrained component has the potential of exceeding 2Gb, the exception NUMERIC_ERROR is raised. Assignment of a default maximum discriminant value does not occur. For example:

```
type DYNAMIC_STRING( LENGTH : NATURAL := 10 )
     is record
     STR : STRING( 1 .. LENGTH );
     end record;
DSTR : DYNAMIC_STRING;
```

For this record, the compiler attempts to allocate NATURAL'LAST bytes for the record. Because this is greater than 2Gb, the exception NUMERIC_ERROR is raised. However, the declaration

```
D : DYNAMIC_STRING(80);
```

raises no exception and creates a record containing an 80-byte string.

F.13 TASKING IMPLEMENTATION

The C^{3}Ada system implements fully pre-emptive and priority-driven tasking. Pre-emptive means that task switches may take place even when the currently running task does not voluntarily give up processor control. This may happen when a task with a high priority is waiting on an external event (the time period specified in a delay statement expires). When this event occurs, processor control is passed to the waiting task immediately if it has the highest priority of the tasks ready to run.

The C^{3}Ada run-time system keeps track of all tasks in two categories: tasks which are ready to run and those that are suspended because they are waiting for something (e.g., a rendezvous to occur or waiting in a delay statement). The tasks ready to run are sorted in a queue by priority (high priorities first). Within one priority, they are sorted in the order in which they entered the "ready" state (tasks waiting longer are served first). Whenever the run-time system needs a task to schedule, the first task in the queue is selected and run.

The accuracy of delay statements is governed by the resolution of the operating system clock which is 1.0/100.00 seconds (or 10ms) in 50 Hz areas and 1.0/120.0 in 60 Hz (SYSTEM.TICK). Although the resolution of the type DURATION is much higher (2^{-14} seconds), task switches caused by the expiration of a delay can only take place on a clock tick. A task waiting in a delay enters the "ready" state when the next clock tick after its delay period has expired.

There is only one OS/32 task for all Ada tasks in this model.

Tasks that depend on library packages continue to execute when the main program terminates. However, when the main program terminates due to an unhandled exception, the tasks will be terminated thereafter.

F.14 NUMERIC ERROR

The predefined exception NUMERIC_ERROR is never raised implicitly by any predefined operation; instead, the predefined exception CONSTRAINT_ERROR is raised.