Byron

Implementation Manual

DRAFT DRAFT DRAFT

Table of Contents

Introduction

Purpose

The purpose of this documentation is threefold: firstly, to document the structure of the Byron compiler; secondly, to document the implementation decisions & limitations; thirdly, to comply with **Annex M** of the *Ada Reference Manual*.

Documentation

This documentation is provided to assist people wishing to understand the design-decisions for the Byron compiler or the principles underlying its construction, modify the compiler, or compare the facilities and capabilities to other compilers.

Structure

Document

This Document is structured [[[]]]

Compiler

The translator is based on several ideas which influence its structure — perhaps the most influential being in the ideal development system, user programs and dependant libraries should not be stored as text residing in the file-system, but rather as meaningful structures held within a database. While such database-powered support environment is arguably outside the scope of the translator proper, as the translator is merely a part of it, the primary internal representation of the translator is designed with such a system in mind.

This design decision is intended to reduce external dependencies (such as whether the OS uses `CRLF`, `LF`, or some other delimiter of text), help with organization and maintenance of the user's projects, as well as improve the possible tooling that might be provided; while the tooling is beyond the scope of the translator proper, it may help the reader to understand the envisioned nature of these tools and how this decision impacts them.

A. Continuous Integration — The paper Workspaces and Experimental Databases: Automated Support for Software Maintenance and Evolution; details a system where the program is held in a hierarchical database, structured along modules and/or 'workspaces' and this structuring encapsulates the functionality of continuous integration and in a better manner: there are no 'non-compilable' states.
B. Project Management — The ability to specify and organize a project is something
C. Package/Library Distribution — Taking (A) and (B) into consideration, a system akin to Perl's CPAN could be constructed, distributing the internally-held DB constructions to the general public thereby ensuring that the Ada community has access to compilable & compatible libraries.

Annex M Compliance

[[[]]]

Compiler

Structure

The translator is structured into various modules, chained together to produce the output from the given input, conceptually as the following subprogram signature details:

`Function Translate( Input : Input_Type ) Return Output_Type`.

This is realized and enforced by the phases of the compiler as represented by the following generic specification:

```
Generic
   Type Input_Type(<>)  is private;
   Type Output_Type(<>) is private;
   with Function "="(Left, Right : Input_Type ) return Boolean is <>;
   with Function "="(Left, Right : Output_Type) return Boolean is <>;
   with Function Translate(Input : Input_Type) return Output_Type;
   Input_Transformation  : access Procedure( Object : in out Input_Type  ):= Null;
   Output_Transformation : access Procedure( Object : in out Output_Type ):= Null;
Function Byron.Generics.Pass( Input : Input_Type ) Return Output_Type;
```

1 — A 'pass' of the compiler.

Virtually all of the main/high-level components of the Byron translator are instantiations of the `Pass` compilation-unit in order to maintain maintainability.

Reader

Purpose

The object of the reader is, primarily, a stopgap. It is meant to handle the common development practices of storing the source code in a text-file, it therefore returns the contents of given text-file as a string.

Operation

The text file-reader automatically converts ASCII, UTF8, UTF16 (both big-endian and little-endian), and UTF32 into a `Wide_Wide_String` (UTF32 code-points) in order to facilitate the handling of Unicode within the source.

Implementation Notes

The entire file is read as a stream of bytes (`String`), and the header of the result is examined for the UTF Byte-Order-Mark for the various forms, and the appropriate translation function is selected — at this point the data is converted into UTF32 and

passed to the rest of the compiler.

Pass

```
Function Byron.Reader is new Byron.Generics.Pass(
  Input_Type      => Internals.Types.Stream_Class,
  Output_Type     => Wide_Wide_String,
  Translate       => Readington
 );
```

Tokenizer

Purpose

The Tokenizer incrementally splits the text into tokens in a process called lexing; in order to do this, there are several intermediate tokens that are used only in the processing. (See Byron Tokens.)

Operation

The Tokenizer takes the provided Wide_Wide_String and outputs a vector of tokens, via the provided process: Byron.Internals.Lexer.Process — which is the successive execution of P1 through P20 in series via the repeated instantiation of the generic procedure in Byron.Internals.SPARK.Element.Transform — which yields a stream of tokens from the Emitable subtype of the Lexington.Aux package.

Implementation Notes

The functionality of the P# procedures are as follows:

1. Generates WHITESPACE.
2. Generates End_Of_Line.
3. Generates Comments on TEXT starting with --.
4. Generates single-character delimeters.
5. Generates double-character delimiters.
6. Generates li_Character for ONLY apostrophe and quote.
7. Generates string literals.
8. Generates Comments.
9. Generates Keywords.
10. Generates Identifiers.
11. Generates Tick.
12. Character literals.
13. Generates integer literals, non-based.
14. Generates float literals, non-based.
15. Generates based integers.
16. Generates based floats.

17. Generates comments passable from the lexer.
18. Generates separators passable from the lexer.
19. Filters out text-artifact tokens.
20. Check for invalid tokens.

At the end of this process, the product is a vector of valid tokens.

Pass

```
Function Byron.Lexer is new Byron.Generics.Pass(
  Input_Type              => Wide_Wide_String,
  Output_Type             => Lexington.Token_Vector_Pkg.Vector,
  Output_Transformation   => Byron.Internals.Lexer.Process'Access,
  Translate               => Lexington.Aux.Po
  );
```

[[TODO: Add more info here]]

Byron Tokens

Natural Tokens

Byron defines 'Natural Tokens' in the Lexington package. These tokens are intended to be used in any tokenizing module and are therefore of a general nature, allocating the entire Natural space for IDs, reserving only the last two values — the penultimate as a general 'text' token, and the last as a 'null' token, symbolizing the end of the token-stream.

Structure

The structure of a token is simply a discriminated record containing a single field of type Wide_Wide_String named Value; the record is discriminated by the ID and the length of the Value field.

A Token is always at least 64-bits, the ID makes up the first 32 bits, while the length is held in the second 32-bits.

```
Type Token( Length, ID : Natural ) is record
        Value : Wide_Wide_String(1..Length);
end record;

For Token use record
        ID      at 0 range 00..31;
        Length  at 0 range 32..63;
end record;
```

4 — The layout and structure of a Token.

Operations

There are five operations defined for the Natural Tokens: Make_Token, Lexeme, Print, the less-than operator ("<"), and the equal operator ("=").

```
Function Make_Token(        ID      : Natural;
                            Value   : Wide_Wide_String
                    ) return    Token;
Function Lexeme     ( Item : Token ) return Wide_Wide_String;
Function Print      ( Item : Token ) return Wide_Wide_String;
Function "<"  ( Left, Right : Token ) return Boolean;
Function "="  ( Left, Right : Token ) return Boolean;
```

5 — Standard operations fot the Token type.

Make_Token

The Make_Token function takes an ID and a Wide_Wide_String and creates a token of the corresponding value.

Lexeme

The Lexeme function returns the data contained in the Value field.

Print

The Print function returns the concatenation of the ID (in textual form), a colon, and the contents of the Value field — this function is intended for use in debugging.

Less-Than

If the IDs are equal then the Values are compared, otherwise the results are the comparisons of the IDs.

(This may have to be replaced with a case-insensitive less-than comparison for the Value field, but as it is now it is a case-sensitive comparison.)

Equal

The equal function only operates on the value of the IDs of given tokens.

Parameterization of Tokens

The package Lexington.Parameters is a Pure generic-package which takes a Token_ID parameter and contains a derivation of the Natural Token type of the Lexington package. The intent here is to take a meaningful enumeration of token-IDs and 'superimpose' them upon the Natural-typed IDs of the Token-type found in Lexington.

Operations

The operations on this specialized-ID Token-type are conversions to- and from- the base Token-type, retrieval of the ID, and a new Print-function.

Conversions

The unary-plus operator "+" renames the private conversion functions publicly for the clients of an instantiation of Lexington.Parameters to use. These are overloaded for both to- and from-conversions.

ID

Given a specialized Token the ID function returns the associated ID of the type provided to the instantiation's parameter.

Print

The Print function as in its parent package is intended for debugging, but is slightly more complicated; its invocation returns the concatenation of: The Natural-ID enclosed in parentheses, Tab, the image of the Token_ID, CRLF, Tab, and the Lexeme enclosed in square-brackets.

Specialized Tokens

For Ada, Byron specializes tokens in the Lexington.Aux package. In this packages the IDs are enumerated, but grouped in offsets of 128 — the only reason that 128 is used is because it is the power-of-two which is immediately after 72, the number of keywords in Ada — these offsets are used purely by convention and are not meaningful to the compiler internals in any way.

Intermediate Representations

DIANA–2012

DIANA, which stands for *Descriptive Intermediate Attributed Notation for Ada*, is an abstract data type [and also an attributed Abstract Syntax Tree] such that the value of every such object is the intermediate form representing a valid Ada program.

The original design, for Ada-83 and available from DITC, was to present a regular, representation independent, efficiently implementable construct based on Ada's formal definition with a single definition for each Ada entity and consideration given to Ada's separate compilation facility — in addition to being updated for the Ada-2012 standard, DIANA–2012 adds one factor to this list: database-amiability.

DIANA was used in several Ada compilers, and is currently used in Oracle's PL/SQL. This makes the transition to database-amiability (both storage and manipulation of the IR) much more attractive, as DIANA is already used in a similar manner already.

ADA INTERMEDIATE REPRESENTATION (AIR)

[[[]]]

GENERAL INTERMEDIATE REPRESENTATION (GIR)

[[[]]]

Annex M

Summary of Documentation Requirements

The Ada language standard allows for certain target machine dependancies in a controlled manner and requires each Ada implementation to document the characteristics and properties of the target system.

This section of the *Implementation Manual* contains the specific documentation requirements mandated by the international standard, including those which are implementation defined and whether the implementation advice was followed.

The following complies with these documentation requirements of Annex M.

M.1 — Specific Documentation Requirements

1/2 In addition to implementation-defined characteristics, each Ada implementation must document various properties of the implementation:

2/2 The behavior of implementations in implementation-defined situations shall be documented — see M.2, "Implementation-Defined Characteristics" for a listing. See 1.1.3(19).

3/2 The set of values that a user-defined Allocate procedure needs to accept for the Alignment parameter. How the standard storage pool is chosen, and how storage is allocated by standard storage pools. See 13.11(22).

4/2 The algorithm used for random number generation, including a description of its period. See A.5.2(44).

5/2 The minimum time interval between calls to the time-dependent Reset procedure that is guaranteed to initiate different random number sequences. See A.5.2(45).

6/2 The conditions under which Io_Exceptions.Name_Error, Io_Exceptions.Use_Error, and Io_Exceptions.Device_Error are propagated. See A.13(15).

7/2 The behavior of package Environment_Variables when environment variables are changed by external mechanisms. See A.17(30/2).

8/2 The overhead of calling machine-code or intrinsic subprograms. See C.1(6).

9/2 The types and attributes used in machine code insertions. See C.1(7).

10/2 The subprogram calling conventions for all supported convention identifiers. See C.1(8/3).

11/2 The mapping between the Link_Name or Ada designator and the external link name. See C.1(9).

12/2 The treatment of interrupts. See C.3(22).

13/2 The metrics for interrupt handlers. See C.3.1(16).

14/3 If the Ceiling_Locking policy is in effect, the default ceiling priority for a protected object that specifies an interrupt handler aspect. See C.3.2(24/3).

15/2 Any circumstances when the elaboration of a preelaborated package causes code to be executed. See C.4(12).

16/2 Whether a partition can be restarted without reloading. See C.4(13).

17/2 The effect of calling Current_Task from an entry body or interrupt handler. See C.7.1(19).

18/2 For package Task_Attributes, limits on the number and size of task attributes, and how to configure any limits. See C.7.2(19).

19/2 The metrics for the Task_Attributes package. See C.7.2(27).

20/2 The details of the configuration used to generate the values of all metrics. See D(2).

21/2 The maximum priority inversion a user task can experience from the implementation. See D.2.3(12/2).

22/2 The amount of time that a task can be preempted for processing on behalf of lower-priority tasks. See D.2.3(13/2).

23/2 The quantum values supported for round robin dispatching. See D.2.5(16/2).

24/2 The accuracy of the detection of the exhaustion of the budget of a task for round robin dispatching. See D.2.5(17/2).

25/2 Any conditions that cause the completion of the setting of the deadline of a task to be delayed for a multiprocessor. See D.2.6(32/2).

26/2 Any conditions that cause the completion of the setting of the priority of a task to be delayed for a multiprocessor. See D.5.1(12.1/2).

27/2 The metrics for Set_Priority. See D.5.1(14).

28/2 The metrics for setting the priority of a protected object. See D.5.2(10).

29/2 On a multiprocessor, any conditions that cause the completion of an aborted construct to be delayed later than what is specified for a single processor. See D.6(3).

30/2 The metrics for aborts. See D.6(8).

31/2 The values of Time_First, Time_Last, Time_Span_First, Time_Span_Last, Time_Span_Unit, and Tick for package Real_Time. See D.8(33).

32/2 The properties of the underlying time base used in package Real_Time. See D.8(34).

33/2 Any synchronization of package Real_Time with external time references. See D.8(35).

34/2 Any aspects of the external environment that could interfere with package Real_Time. See D.8(36/3).

35/2 The metrics for package Real_Time. See D.8(45).

36/2 The minimum value of the delay expression of a delay_relative_statement that causes a task to actually be blocked. See D.9(7).

37/2 The minimum difference between the value of the delay expression of a delay_until_statement and the value of Real_Time.Clock, that causes the task to actually be blocked. See D.9(8).

38/2 The metrics for delay statements. See D.9(13).

39/2 The upper bound on the duration of interrupt blocking caused by the implementation. See D.12(5).

40/2 The metrics for entry-less protected objects. See D.12(12).

41/2 The values of CPU_Time_First, CPU_Time_Last, CPU_Time_Unit, and CPU_Tick of package Execution_Time. See D.14(21/2).

42/3 The properties of the mechanism used to implement package Execution_Time, including the values of the constants defined in the package. See D.14(22/2).

43/2 The metrics for execution time. See D.14(27).

44/2 The metrics for timing events. See D.15(24).

44.1/3 The processor(s) on which the clock interrupt is handled; the processors on which each Interrupt_Id can be handled. See D.16.1(32).

45/2 Whether the RPC-receiver is invoked from concurrent tasks, and if so, the number of such tasks. See E.5(25).

46/2 Any techniques used to reduce cancellation errors in Numerics.Generic_Real_Arrays shall be documented. See G.3.1(86/2).

47/2 Any techniques used to reduce cancellation errors in Numerics.Generic_Complex_Arrays shall be documented. See G.3.2(155/2).

48/2 If a pragma Normalize_Scalars applies, the implicit initial values of scalar subtypes shall be documented. Such a value should be an invalid representation when possible; any cases when is it not shall be documented. See H.1(5/2).

49/2 The range of effects for each bounded error and each unspecified effect. If the effects of a given erroneous construct are constrained, the constraints shall be documented. See H.2(1).

50/2 For each inspection point, a mapping between each inspectable object and the machine resources where the object's value can be obtained shall be provided. See H.3.2(8).

51/2 If a pragma Restrictions(No_Exceptions) is specified, the effects of all constructs where language-defined checks are still performed. See H.4(25).

52/2 The interrupts to which a task entry may be attached. See J.7.1(12).

53/2 The type of entry call invoked for an interrupt entry. See J.7.1(13).

M.2 — Implementation-Defined Characteristics

1/2 The Ada language allows for certain machine dependences in a controlled manner. Each Ada implementation must document all implementation-defined characteristics:

2/2 Whether or not each recommendation given in Implementation Advice is followed — see M.3, “Implementation Advice” for a listing. See 1.1.2(37).

3 Capacity limitations of the implementation. See 1.1.3(3).

4 Variations from the standard that are impractical to avoid given the implementation's execution environment. See 1.1.3(6).

5 Which code_statements cause external interactions. See 1.1.3(10).

6 The coded representation for the text of an Ada program. See 2.1(4/3).

6.1/2 The semantics of an Ada program whose text is not in Normalization Form KC. See 2.1(4.1/3).

7/2 This paragraph was deleted.

8 The representation for an end of line. See 2.2(2/3).

9 Maximum supported line length and lexical element length. See 2.2(14).

10 Implementation-defined pragmas. See 2.8(14).

11 Effect of pragma Optimize. See 2.8(27).

11.1/4 The message string associated with the Assertion_Error exception raised by the

failure of a predicate check if there is no applicable Predicate_Failure aspect. See 3.2.4(31).

11.2/2 The sequence of characters of the value returned by S'Wide_Image when some of the graphic characters of S'Wide_Wide_Image are not defined in Wide_Character. See 3.5(30/3).

12/2 The sequence of characters of the value returned by S'Image when some of the graphic characters of S'Wide_Wide_Image are not defined in Character. See 3.5(37/3).

13 The predefined integer types declared in Standard. See 3.5.4(25).

14 Any nonstandard integer types and the operators defined for them. See 3.5.4(26).

15 Any nonstandard real types and the operators defined for them. See 3.5.6(8).

16 What combinations of requested decimal precision and range are supported for floating point types. See 3.5.7(7).

17 The predefined floating point types declared in Standard. See 3.5.7(16).

18 The small of an ordinary fixed point type. See 3.5.9(8/2).

19 What combinations of small, range, and digits are supported for fixed point types. See 3.5.9(10).

20/2 The result of Tags.Wide_Wide_Expanded_Name for types declared within an unnamed block_statement. See 3.9(10).

20.1/2 The sequence of characters of the value returned by Tags.Expanded_Name (respectively, Tags.Wide_Expanded_Name) when some of the graphic characters of Tags.Wide_Wide_Expanded_Name are not defined in Character (respectively, Wide_Character). See 3.9(10.1/2).

21 Implementation-defined attributes. See 4.1.4(12/1).

21.1/2 Rounding of real static expressions which are exactly half-way between two machine numbers. See 4.9(38/2).

22 Any implementation-defined time types. See 9.6(6/3).

23 The time base associated with relative delays. See 9.6(20).

24 The time base of the type Calendar.Time. See 9.6(23).

25/2 The time zone used for package Calendar operations. See 9.6(24/2).

26 Any limit on delay_until_statements of select_statements. See 9.6(29).

26.1/2 The result of Calendar.Formating.Image if its argument represents more than 100 hours. See 9.6.1(86/2).

27/3 This paragraph was deleted.

28 The representation for a compilation. See 10.1(2).

29 Any restrictions on compilations that contain multiple compilation_units. See 10.1(4).

30 The mechanisms for creating an environment and for adding and replacing compilation units. See 10.1.4(3/2).

30.1/2 The mechanisms for adding a compilation unit mentioned in a limited_with_clause to an environment. See 10.1.4(3/2).

31 The manner of explicitly assigning library units to a partition. See 10.2(2).

32 The implementation-defined means, if any, of specifying which compilation units are needed by a given compilation unit. See 10.2(2).

33 The manner of designating the main subprogram of a partition. See 10.2(7).
34 The order of elaboration of library_items. See 10.2(18).
35 Parameter passing and function return for the main subprogram. See 10.2(21).
36 The mechanisms for building and running partitions. See 10.2(24).
37 The details of program execution, including program termination. See 10.2(25).
38 The semantics of any nonactive partitions supported by the implementation. See 10.2(28/3).
39 The information returned by Exception_Message. See 11.4.1(10.1/4).
40/2 The result of Exceptions.Wide_Wide_Exception_Name for exceptions declared within an unnamed block_statement. See 11.4.1(12).
40.1/2 The sequence of characters of the value returned by Exceptions.Exception_Name (respectively, Exceptions.Wide_Exception_Name) when some of the graphic characters of Exceptions.Wide_Wide_Exception_Name are not defined in Character (respectively, Wide_Character). See 11.4.1(12.1/2).
41 The information returned by Exception_Information. See 11.4.1(13/2).
41.1/3 Implementation-defined policy_identifiers and assertion_aspect_marks allowed in a pragma Assertion_Policy. See 11.4.2(9/3).
41.2/2 The default assertion policy. See 11.4.2(10).
42 Implementation-defined check names. See 11.5(27).
42.1/2 Existence and meaning of second parameter of pragma Unsuppress. See 11.5(27.1/2).
42.2/2 The cases that cause conflicts between the representation of the ancestors of a type_declaration. See 13.1(13.1/3).
43/3 The interpretation of each representation aspect. See 13.1(20).
44/3 Any restrictions placed upon the specification of representation aspects. See 13.1(20).
44.1/3 Implementation-defined aspects, inluding the syntax for specifying such aspects and the legality rules for such aspects. See 13.1.1(38).
44.2/2 The set of machine scalars. See 13.3(8.1/3).
45 The meaning of Size for indefinite subtypes. See 13.3(48).
46 The default external representation for a type tag. See 13.3(75/3).
47 What determines whether a compilation unit is the same in two different partitions. See 13.3(76).
48 Implementation-defined components. See 13.5.1(15).
49 If Word_Size = Storage_Unit, the default bit ordering. See 13.5.3(5).
50/2 The contents of the visible part of package System. See 13.7(2).
50.1/2 The range of Storage_Elements.Storage_Offset, the modulus of Storage_Elements.Storage_Element, and the declaration of Storage_Elements.Integer_Address.. See 13.7.1(11).
51 The contents of the visible part of package System.Machine_Code, and the meaning of code_statements. See 13.8(7).
51.1/2 The result of unchecked conversion for instances with scalar result types whose result is not defined by the language. See 13.9(11).
52/2 The effect of unchecked conversion for instances with nonscalar result types

whose effect is not defined by the language. See 13.9(11).

53/2 This paragraph was deleted.

54 Whether or not the implementation provides user-accessible names for the standard pool type(s). See 13.11(17).

55/2 The meaning of Storage_Size when neither the Storage_Size nor the Storage_Pool is specified for an access type. See 13.11(18).

56/2 This paragraph was deleted.

56.1/4 The effect of specifying aspect Default_Storage_Pool on an instance of a language-defined generic unit. See 13.11.3(5).

57/3 This paragraph was deleted.

57.1/3 Implementation-defined restrictions allowed in a pragma Restrictions. See 13.12(8.7/3).

58 The consequences of violating limitations on Restrictions pragmas. See 13.12(9).

58.1/3 Implementation-defined usage profiles allowed in a pragma Profile. See 13.12(15).

59/2 The contents of the stream elements read and written by the Read and Write attributes of elementary types. See 13.13.2(9).

60 The names and characteristics of the numeric subtypes declared in the visible part of package Standard. See A.1(3).

60.1/2 The values returned by Strings.Hash. See A.4.9(3/2).

61 The accuracy actually achieved by the elementary functions. See A.5.1(1).

62 The sign of a zero result from some of the operators or functions in Numerics.Generic_Elementary_Functions, when Float_Type'Signed_Zeros is True. See A.5.1(46).

63 The value of Numerics.Float_Random.Max_Image_Width. See A.5.2(27).

64 The value of Numerics.Discrete_Random.Max_Image_Width. See A.5.2(27).

65/2 This paragraph was deleted.

66 The string representation of a random number generator's state. See A.5.2(38).

67/2 This paragraph was deleted.

68 The values of the Model_Mantissa, Model_Emin, Model_Epsilon, Model, Safe_First, and Safe_Last attributes, if the Numerics Annex is not supported. See A.5.3(72).

69/2 This paragraph was deleted.

70 The value of Buffer_Size in Storage_IO. See A.9(10).

71/2 The external files associated with the standard input, standard output, and standard error files. See A.10(5).

72 The accuracy of the value produced by Put. See A.10.9(36).

72.1/1 Current size for a stream file for which positioning is not supported. See A.12.1(1.1/1).

73/2 The meaning of Argument_Count, Argument, and Command_Name for package Command_Line. The bounds of type Command_Line.Exit_Status. See A.15(1).

73.1/2 The interpretation of file names and directory names. See A.16(46/2).

73.2/2 The maximum value for a file size in Directories. See A.16(87/2).

73.3/2 The result for Directories.Size for a directory or special file See A.16(93/2).

73.4/2 The result for Directories.Modification_Time for a directory or special file. See

A.16(95/2).

73.5/2 The interpretation of a nonnull search pattern in Directories. See A.16(104/3).

73.6/2 The results of a Directories search if the contents of the directory are altered while a search is in progress. See A.16(110/3).

73.7/2 The definition and meaning of an environment variable. See A.17(1/2).

73.8/2 The circumstances where an environment variable cannot be defined. See A.17(16/2).

73.9/2 Environment names for which Set has the effect of Clear. See A.17(17/2).

73.10/2 The value of Containers.Hash_Type'Modulus. The value of Containers.Count_Type'Last. See A.18.1(7/2).

74 Implementation-defined convention names. See B.1(11/3).

75 The meaning of link names. See B.1(36).

76 The manner of choosing link names when neither the link name nor the address of an imported or exported entity is specified. See B.1(36).

77 The effect of pragma Linker_Options. See B.1(37).

78 The contents of the visible part of package Interfaces and its language-defined descendants. See B.2(1).

79/2 Implementation-defined children of package Interfaces. See B.2(11).

79.1/2 The definitions of certain types and constants in Interfaces.C. See B.3(41).

80/1 The types Floating, Long_Floating, Binary, Long_Binary, Decimal_Element, and COBOL_Character; and the initializations of the variables Ada_To_COBOL and COBOL_To_Ada, in Interfaces.COBOL. See B.4(50).

80.1/1 The types Fortran_Integer, Real, Double_Precision, and Character_Set in Interfaces.Fortran. See B.5(17).

81/2 Implementation-defined intrinsic subprograms. See C.1(1/3).

82/2 This paragraph was deleted.

83/2 This paragraph was deleted.

83.1/3 Any restrictions on a protected procedure or its containing type when an aspect Attach_handler or Interrupt_Handler is specified. See C.3.1(17).

83.2/3 Any other forms of interrupt handler supported by the Attach_Handler and Interrupt_Handler aspects. See C.3.1(19).

84/2 This paragraph was deleted.

85/4 The semantics of some attributes and functions of an entity for which aspect Discard_Names is True. See C.5(7).

86 The result of the Task_Identification.Image attribute. See C.7.1(7).

87/2 The value of Current_Task when in a protected entry, interrupt handler, or finalization of a task attribute. See C.7.1(17/3).

88/2 This paragraph was deleted.

88.1/1 Granularity of locking for Task_Attributes. See C.7.2(16/1).

89/2 This paragraph was deleted.

90/2 This paragraph was deleted.

91 The declarations of Any_Priority and Priority. See D.1(11).

92 Implementation-defined execution resources. See D.1(15).

93 Whether, on a multiprocessor, a task that is waiting for access to a protected

object keeps its processor busy. See D.2.1(3).

94/2 The effect of implementation-defined execution resources on task dispatching. See D.2.1(9/2).

95/2 This paragraph was deleted.

96/2 This paragraph was deleted.

97/2 Implementation defined task dispatching policies. See D.2.2(19).

97.1/2 The value of Default_Quantum in Dispatching.Round_Robin. See D.2.5(4).

98 Implementation-defined policy_identifiers allowed in a pragma Locking_Policy. See D.3(4).

98.1/2 The locking policy if no Locking_Policy pragma applies to any unit of a partition. See D.3(6).

99 Default ceiling priorities. See D.3(10/4).

100 The ceiling of any protected object used internally by the implementation. See D.3(16).

101 Implementation-defined queuing policies. See D.4(1/3).

102/2 This paragraph was deleted.

103 Any operations that implicitly require heap storage allocation. See D.7(8).

103.1/4 When restriction No_Dynamic_CPU_Assignment applies to a partition, the processor on which a task with a CPU value of a Not_A_Specific_CPU will execute. See D.7(10).

103.2/2 When restriction No_Task_Termination applies to a partition, what happens when a task terminates. See D.7(15.1/2).

103.3/2 The behavior when restriction Max_Storage_At_Blocking is violated. See D.7(17/1).

103.4/2 The behavior when restriction Max_Asynchronous_Select_Nesting is violated. See D.7(18/1).

103.5/2 The behavior when restriction Max_Tasks is violated. See D.7(19).

104/2 Whether the use of pragma Restrictions results in a reduction in program code or data size or execution time. See D.7(20).

105/2 This paragraph was deleted.

106/2 This paragraph was deleted.

106.1/3 The value of Barrier_Limit'Last in Synchronous_Barriers. See D.10.1(4/3).

106.2/3 When an aborted task that is waiting on a Synchronous_Barrier is aborted. See D.10.1(13/3).

107/2 This paragraph was deleted.

107.1/3 The value of Min_Handler_Ceiling in Execution_Time.Group_Budgets. See D.14.2(7/2).

107.2/3 The value of CPU_Range'Last in System.Multiprocessors. See D.16(4/3).

107.3/3 The processor on which the environment task executes in the absence of a value for the aspect CPU. See D.16(13/3).

108 The means for creating and executing distributed programs. See E(5).

109 Any events that can result in a partition becoming inaccessible. See E.1(7).

110 The scheduling policies, treatment of priorities, and management of shared resources between partitions in certain cases. See E.1(11).

111/1 This paragraph was deleted.

112 Whether the execution of the remote subprogram is immediately aborted as a result of cancellation. See E.4(13).

112.1/2 The range of type System.RPC.Partition_Id. See E.5(14).

113/2 This paragraph was deleted.

114 Implementation-defined interfaces in the PCS. See E.5(26).

115 The values of named numbers in the package Decimal. See F.2(7).

116 The value of Max_Picture_Length in the package Text_IO.Editing See F.3.3(16).

117 The value of Max_Picture_Length in the package Wide_Text_IO.Editing See F.3.4(5).

117.1/2 The value of Max_Picture_Length in the package Wide_Wide_Text_IO.Editing See F.3.5(5).

118 The accuracy actually achieved by the complex elementary functions and by other complex arithmetic operations. See G.1(1).

119 The sign of a zero result (or a component thereof) from any operator or function in Numerics.Generic_Complex_Types, when Real'Signed_Zeros is True. See G.1.1(53).

120 The sign of a zero result (or a component thereof) from any operator or function in Numerics.Generic_Complex_Elementary_Functions, when Complex_Types.Real'Signed_Zeros is True. See G.1.2(45).

121 Whether the strict mode or the relaxed mode is the default. See G.2(2).

122 The result interval in certain cases of fixed-to-float conversion. See G.2.1(10).

123 The result of a floating point arithmetic operation in overflow situations, when the Machine_Overflows attribute of the result type is False. See G.2.1(13).

124 The result interval for division (or exponentiation by a negative exponent), when the floating point hardware implements division as multiplication by a reciprocal. See G.2.1(16).

125 The definition of close result set, which determines the accuracy of certain fixed point multiplications and divisions. See G.2.3(5).

126 Conditions on a universal_real operand of a fixed point multiplication or division for which the result shall be in the perfect result set. See G.2.3(22).

127 The result of a fixed point arithmetic operation in overflow situations, when the Machine_Overflows attribute of the result type is False. See G.2.3(27).

128 The result of an elementary function reference in overflow situations, when the Machine_Overflows attribute of the result type is False. See G.2.4(4).

129 The value of the angle threshold, within which certain elementary functions, complex arithmetic operations, and complex elementary functions yield results conforming to a maximum relative error bound. See G.2.4(10).

130 The accuracy of certain elementary functions for parameters beyond the angle threshold. See G.2.4(10).

131 The result of a complex arithmetic operation or complex elementary function reference in overflow situations, when the Machine_Overflows attribute of the corresponding real type is False. See G.2.6(5).

132 The accuracy of certain complex arithmetic operations and certain complex elementary functions for parameters (or components thereof) beyond the angle

threshold. See G.2.6(8).

132.1/2 The accuracy requirements for the subprograms Solve, Inverse, Determinant, Eigenvalues and Eigensystem for type Real_Matrix. See G.3.1(81/2).

132.2/2 The accuracy requirements for the subprograms Solve, Inverse, Determinant, Eigenvalues and Eigensystem for type Complex_Matrix. See G.3.2(149/2).

133/2 This paragraph was deleted.

134/2 This paragraph was deleted.

135/2 This paragraph was deleted.

136/2 This paragraph was deleted.

136.1/2 Implementation-defined policy_identifiers allowed in a pragma Partition_Elaboration_Policy. See H.6(4/2).

M.3 — Implementation Advice

1/2 This International Standard sometimes gives advice about handling certain target machine dependences. Each Ada implementation must document whether that advice is followed:

2/2 Program_Error should be raised when an unsupported Specialized Needs Annex feature is used at run time. See 1.1.3(20).

3/2 Implementation-defined extensions to the functionality of a language-defined library unit should be provided by adding children to the library unit. See 1.1.3(21).

4/2 If a bounded error or erroneous execution is detected, Program_Error should be raised. See 1.1.5(12).

5/2 Implementation-defined pragmas should have no semantic effect for error-free programs. See 2.8(16/3).

6/2 Implementation-defined pragmas should not make an illegal program legal, unless they complete a declaration or configure the library_items in an environment. See 2.8(19).

7/2 Long_Integer should be declared in Standard if the target supports 32-bit arithmetic. No other named integer subtypes should be declared in Standard. See 3.5.4(28).

8/2 For a two's complement target, modular types with a binary modulus up to System.Max_Int*2+2 should be supported. A nonbinary modulus up to Integer'Last should be supported. See 3.5.4(29).

9/2 Program_Error should be raised for the evaluation of S'Pos for an enumeration type, if the value of the operand does not correspond to the internal code for any enumeration literal of the type. See 3.5.5(8).

10/2 Long_Float should be declared in Standard if the target supports 11 or more digits of precision. No other named float subtypes should be declared in Standard. See 3.5.7(17).

11/2 Multidimensional arrays should be represented in row-major order, unless the array has convention Fortran. See 3.6.2(11/3).

12/3 Tags.Internal_Tag should return the tag of a type, if one exists, whose innermost

master is a master of the point of the function call.. See 3.9(26.1/3).

13/2 A real static expression with a nonformal type that is not part of a larger static expression should be rounded the same as the target system. See 4.9(38.1/2).

14/2 The value of Duration'Small should be no greater than 100 microseconds. See 9.6(30).

15/2 The time base for delay_relative_statements should be monotonic. See 9.6(31).

16/2 Leap seconds should be supported if the target system supports them. Otherwise, operations in Calendar.Formatting should return results consistent with no leap seconds. See 9.6.1(89/2).

17/2 When applied to a generic unit, a program unit pragma that is not a library unit pragma should apply to each instance of the generic unit for which there is not an overriding pragma applied directly to the instance. See 10.1.5(10/1).

18/2 A type declared in a preelaborated package should have the same representation in every elaboration of a given version of the package. See 10.2.1(12).

19/2 Exception_Information should provide information useful for debugging, and should include the Exception_Name and Exception_Message. See 11.4.1(19).

20/2 Exception_Message by default should be short, provide information useful for debugging, and should not include the Exception_Name. See 11.4.1(19).

21/2 Code executed for checks that have been suppressed should be minimized. See 11.5(28).

22/2 The recommended level of support for all representation items should be followed. See 13.1(28/3).

23/2 Storage allocated to objects of a packed type should be minimized. See 13.2(6).

24/3 The recommended level of support for the Pack aspect should be followed. See 13.2(9).

25/2 For an array X, X'Address should point at the first component of the array rather than the array bounds. See 13.3(14).

26/2 The recommended level of support for the Address attribute should be followed. See 13.3(19).

26.1/3 For any tagged specific subtype S, S'Class'Alignment should equal S'Alignment. See 13.3(28).

27/2 The recommended level of support for the Alignment attribute should be followed. See 13.3(35).

28/2 The Size of an array object should not include its bounds. See 13.3(41.1/2).

29/2 If the Size of a subtype allows for efficient independent addressability, then the Size of most objects of the subtype should equal the Size of the subtype. See 13.3(52).

30/2 A Size clause on a composite subtype should not affect the internal layout of components. See 13.3(53).

31/2 The recommended level of support for the Size attribute should be followed. See 13.3(56).

32/2 The recommended level of support for the Component_Size attribute should be followed. See 13.3(73).

33/2 The recommended level of support for enumeration_representation_clauses

should be followed. See 13.4(10).

34/2 The recommended level of support for record_representation_clauses should be followed. See 13.5.1(22).

35/2 If a component is represented using a pointer to the actual data of the component which is contiguous with the rest of the object, then the storage place attributes should reflect the place of the actual data. If a component is allocated discontiguously from the rest of the object, then a warning should be generated upon reference to one of its storage place attributes. See 13.5.2(5).

36/2 The recommended level of support for the nondefault bit ordering should be followed. See 13.5.3(8).

37/2 Type System.Address should be a private type. See 13.7(37).

38/2 Operations in System and its children should reflect the target environment; operations that do not make sense should raise Program_Error. See 13.7.1(16).

39/2 Since the Size of an array object generally does not include its bounds, the bounds should not be part of the converted data in an instance of Unchecked_Conversion. See 13.9(14/2).

40/2 There should not be unnecessary run-time checks on the result of an Unchecked_Conversion; the result should be returned by reference when possible. Restrictions on Unchecked_Conversions should be avoided. See 13.9(15).

41/2 The recommended level of support for Unchecked_Conversion should be followed. See 13.9(17).

42/2 Any cases in which heap storage is dynamically allocated other than as part of the evaluation of an allocator should be documented. See 13.11(23).

43/2 A default storage pool for an access-to-constant type should not have overhead to support deallocation of individual objects. See 13.11(24).

44/2 Usually, a storage pool for an access discriminant or access parameter should be created at the point of an allocator, and be reclaimed when the designated object becomes inaccessible. For other anonymous access types, the pool should be created at the point where the type is elaborated and need not support deallocation of individual objects. See 13.11(25).

45/2 For a standard storage pool, an instance of Unchecked_Deallocation should actually reclaim the storage. See 13.11.2(17).

45.1/3 A call on an instance of Unchecked_Deallocation with a nonnull access value should raise Program_Error if the actual access type of the instance is a type for which the Storage_Size has been specified to be zero or is defined by the language to be zero. See 13.11.2(17.1/3).

46/2 If not specified, the value of Stream_Size for an elementary type should be the number of bits that corresponds to the minimum number of stream elements required by the first subtype of the type, rounded up to the nearest factor or multiple of the word size that is also a multiple of the stream element size. See 13.13.2(1.6/2).

47/2 The recommended level of support for the Stream_Size attribute should be followed. See 13.13.2(1.8/2).

48/2 If an implementation provides additional named predefined integer types, then the

names should end with “Integer”. If an implementation provides additional named predefined floating point types, then the names should end with “Float”. See A.1(52).

49/2 Implementation-defined operations on Wide_Character, Wide_String, Wide_Wide_Character, and Wide_Wide_String should be child units of Wide_Characters or Wide_Wide_Characters. See A.3.1(7/3).

49.1/3 The string returned by Wide_Characters.Handling.Character_Set_Version should include either “10646:” or “Unicode”. See A.3.5(62).

50/2 Bounded string objects should not be implemented by implicit pointers and dynamic allocation. See A.4.4(106).

51/2 Strings.Hash should be good a hash function, returning a wide spread of values for different string values, and similar strings should rarely return the same value. See A.4.9(12/2).

51.1/3 If an implementation supports other string encoding schemes, a child of Ada.Strings similar to UTF_Encoding should be defined. See A.4.11(107/3).

52/2 Any storage associated with an object of type Generator of the random number packages should be reclaimed on exit from the scope of the object. See A.5.2(46).

53/2 Each value of Initiator passed to Reset for the random number packages should initiate a distinct sequence of random numbers, or, if that is not possible, be at least a rapidly varying function of the initiator value. See A.5.2(47).

54/2 Get_Immediate should be implemented with unbuffered input; input should be available immediately; line-editing should be disabled. See A.10.7(23).

55/2 Package Directories.Information should be provided to retrieve other information about a file. See A.16(124/2).

56/3 Directories.Start_Search and Directories.Search should raise Name_Error for malformed patterns. See A.16(125).

57/2 Directories.Rename should be supported at least when both New_Name and Old_Name are simple names and New_Name does not identify an existing external file. See A.16(126/2).

57.1/3 Directories.Hierarchical_File_Names should be provided for systems with hierarchical file naming, and should not be provided on other systems. See A.16.1(36/3).

58/2 If the execution environment supports subprocesses, the current environment variables should be used to initialize the environment variables of a subprocess. See A.17(32/2).

59/2 Changes to the environment variables made outside the control of Environment_Variables should be reflected immediately. See A.17(33/2).

60/2 Containers.Hash_Type'Modulus should be at least 2**32. Containers.Count_Type'Last should be at least 2**31–1. See A.18.1(8/2).

61/2 The worst-case time complexity of Element for Containers.Vector should be O(log N). See A.18.2(256/2).

62/2 The worst-case time complexity of Append with Count = 1 when N is less than the capacity for Containers.Vector should be O(log N). See A.18.2(257/2).

63/2 The worst-case time complexity of Prepend with Count = 1 and Delete_First with

Count=1 for Containers.Vectors should be O(N log N). See A.18.2(258/2).

64/2 The worst-case time complexity of a call on procedure Sort of an instance of Containers.Vectors.Generic_Sorting should be O(N**2), and the average time complexity should be better than O(N**2). See A.18.2(259/2).

65/2 Containers.Vectors.Generic_Sorting.Sort and Containers.Vectors.Generic_Sorting.Merge should minimize copying of elements. See A.18.2(260/2).

66/2 Containers.Vectors.Move should not copy elements, and should minimize copying of internal data structures. See A.18.2(261/2).

67/2 If an exception is propagated from a vector operation, no storage should be lost, nor any elements removed from a vector unless specified by the operation. See A.18.2(262/2).

68/2 The worst-case time complexity of Element, Insert with Count=1, and Delete with Count=1 for Containers.Doubly_Linked_Lists should be O(log N). See A.18.3(160/2).

69/2 A call on procedure Sort of an instance of Containers.Doubly_Linked_Lists.Generic_Sorting should have an average time complexity better than O(N**2) and worst case no worse than O(N**2). See A.18.3(161/2).

70/2 Containers.Doubly_Linked_Lists.Move should not copy elements, and should minimize copying of internal data structures. See A.18.3(162/2).

71/2 If an exception is propagated from a list operation, no storage should be lost, nor any elements removed from a list unless specified by the operation. See A.18.3(163/2).

72/2 Move for a map should not copy elements, and should minimize copying of internal data structures. See A.18.4(83/2).

73/2 If an exception is propagated from a map operation, no storage should be lost, nor any elements removed from a map unless specified by the operation. See A.18.4(84/2).

74/2 The average time complexity of Element, Insert, Include, Replace, Delete, Exclude and Find operations that take a key parameter for Containers.Hashed_Maps should be O(log N). The average time complexity of the subprograms of Containers.Hashed_Maps that take a cursor parameter should be O(1). The average time complexity of Containers.Hashed_Maps.Reserve_Capacity should be O(N). See A.18.5(62/2).

75/2 The worst-case time complexity of Element, Insert, Include, Replace, Delete, Exclude and Find operations that take a key parameter for Containers.Ordered_Maps should be O((log N)**2) or better. The worst-case time complexity of the subprograms of Containers.Ordered_Maps that take a cursor parameter should be O(1). See A.18.6(95/2).

76/2 Move for sets should not copy elements, and should minimize copying of internal data structures. See A.18.7(104/2).

77/2 If an exception is propagated from a set operation, no storage should be lost, nor any elements removed from a set unless specified by the operation. See

A.18.7(105/2).

78/2 The average time complexity of the Insert, Include, Replace, Delete, Exclude and Find operations of Containers.Hashed_Sets that take an element parameter should be O(log N). The average time complexity of the subprograms of Containers.Hashed_Sets that take a cursor parameter should be O(1). The average time complexity of Containers.Hashed_Sets.Reserve_Capacity should be O(N). See A.18.8(88/2).

79/2 The worst-case time complexity of the Insert, Include, Replace, Delete, Exclude and Find operations of Containers.Ordered_Sets that take an element parameter should be O((log N)**2). The worst-case time complexity of the subprograms of Containers.Ordered_Sets that take a cursor parameter should be O(1). See A.18.9(116/2).

79.1/3 The worst-case time complexity of the Element, Parent, First_Child, Last_Child, Next_Sibling, Previous_Sibling, Insert_Child with Count=1, and Delete operations of Containers.Multiway_Trees should be O(log N). See A.18.10(231/3).

79.2/3 Containers.Multiway_Trees.Move should not copy elements, and should minimize copying of internal data structures. See A.18.10(232/3).

79.3/3 If an exception is propagated from a tree operation, no storage should be lost, nor any elements removed from a tree unless specified by the operation. See A.18.10(233/3).

79.4/3 Containers.Indefinite_Holders.Move should not copy the element, and should minimize copying of internal data structures. See A.18.18(73/3).

79.5/3 If an exception is propagated from a holder operation, no storage should be lost, nor should the element be removed from a holder container unless specified by the operation. See A.18.18(74/3).

79.6/3 Bounded vector objects should be implemented without implicit pointers or dynamic allocation. See A.18.19(16/3).

79.7/3 The implementation advice for procedure Move to minimize copying does not apply to bounded vectors. See A.18.19(17/3).

79.8/3 Bounded list objects should be implemented without implicit pointers or dynamic allocation. See A.18.20(19/3).

79.9/3 The implementation advice for procedure Move to minimize copying does not apply to bounded lists. See A.18.20(20/3).

79.10/3 Bounded hashed map objects should be implemented without implicit pointers or dynamic allocation. See A.18.21(21/3).

79.11/3 The implementation advice for procedure Move to minimize copying does not apply to bounded hashed maps. See A.18.21(22/3).

79.12/3 Bounded ordered map objects should be implemented without implicit pointers or dynamic allocation. See A.18.22(18/3).

79.13/3 The implementation advice for procedure Move to minimize copying does not apply to bounded ordered maps. See A.18.22(19/3).

79.14/3 Bounded hashed set objects should be implemented without implicit pointers or dynamic allocation. See A.18.23(20/3).

79.15/3 The implementation advice for procedure Move to minimize copying does

not apply to bounded hashed sets. See A.18.23(21/3).

79.16/3 Bounded ordered set objects should be implemented without implicit pointers or dynamic allocation. See A.18.24(17/3).

79.17/3 The implementation advice for procedure Move to minimize copying does not apply to bounded ordered sets. See A.18.24(18/3).

79.18/3 Bounded tree objects should be implemented without implicit pointers or dynamic allocation. See A.18.25(19/3).

79.19/3 The implementation advice for procedure Move to minimize copying does not apply to bounded trees. See A.18.25(20/3).

80/2 Containers.Generic_Array_Sort and Containers.Generic_Constrained_Array_Sort should have an average time complexity better than O(N**2) and worst case no worse than O(N**2). See A.18.26(10/2).

81/2 Containers.Generic_Array_Sort and Containers.Generic_Constrained_Array_Sort should minimize copying of elements. See A.18.26(11/2).

81.1/3 Containers.Generic_Sort should have an average time complexity better than O(N**2) and worst case no worse than O(N**2). See A.18.26(12/3).

81.2/3 Containers.Generic_Sort should minimize calls to the generic formal Swap. See A.18.26(13/3).

81.3/3 Bounded queue objects should be implemented without implicit pointers or dynamic allocation. See A.18.29(13/3).

81.4/3 Bounded priority queue objects should be implemented without implicit pointers or dynamic allocation. See A.18.31(14/3).

82/3 If Export is supported for a language, the main program should be able to be written in that language. Subprograms named "adainit" and "adafinal" should be provided for elaboration and finalization of the environment task. See B.1(39/3).

83/3 Automatic elaboration of preelaborated packages should be provided when specifying the Export aspect as True is supported. See B.1(40/3).

84/3 For each supported convention L other than Intrinsic, specifying the aspects Import and Export should be supported for objects of L-compatible types and for subprograms, and aspect Convention should be supported for L-eligible types and for subprograms. See B.1(41/4).

85/2 If an interface to C, COBOL, or Fortran is provided, the corresponding package or packages described in Annex B, “Interface to Other Languages” should also be provided. See B.2(13/3).

86/2 The constants nul, wide_nul, char16_nul, and char32_nul in package Interfaces.C should have a representation of zero. See B.3(62.5/3).

87/2 If C interfacing is supported, the interface correspondences between Ada and C should be supported. See B.3(71).

88/2 If COBOL interfacing is supported, the interface correspondences between Ada and COBOL should be supported. See B.4(98).

89/2 If Fortran interfacing is supported, the interface correspondences between Ada and Fortran should be supported. See B.5(26).

90/2 The machine code or intrinsics support should allow access to all operations normally available to assembly language programmers for the target environment.

See C.1(3).

91/2 Interface to assembler should be supported; the default assembler should be associated with the convention identifier Assembler. See C.1(4/3).

92/2 If an entity is exported to assembly language, then the implementation should allocate it at an addressable location even if not otherwise referenced from the Ada code. A call to a machine code or assembler subprogram should be treated as if it could read or update every object that is specified as exported. See C.1(5).

93/2 Little or no overhead should be associated with calling intrinsic and machine-code subprograms. See C.1(10).

94/2 Intrinsic subprograms should be provided to access any machine operations that provide special capabilities or efficiency not normally available. See C.1(16).

95/2 If the Ceiling_Locking policy is not in effect and the target system allows for finer-grained control of interrupt blocking, a means for the application to specify which interrupts are to be blocked during protected actions should be provided. See C.3(28/2).

96/2 Interrupt handlers should be called directly by the hardware. See C.3.1(20).

97/2 Violations of any implementation-defined restrictions on interrupt handlers should be detected before run time. See C.3.1(21).

98/2 If implementation-defined forms of interrupt handler procedures are supported, then for each such form of a handler, a type analogous to Parameterless_Handler should be specified in a child package of Interrupts, with the same operations as in the predefined package Interrupts. See C.3.2(25).

99/2 Preelaborated packages should be implemented such that little or no code is executed at run time for the elaboration of entities. See C.4(14).

100/4 If aspect Discard_Names is True for an entity, then the amount of storage used for storing names associated with that entity should be reduced. See C.5(8/4).

101/2 A load or store of a volatile object whose size is a multiple of System.Storage_Unit and whose alignment is nonzero, should be implemented by accessing exactly the bits of the object and no others. See C.6(22/2).

102/2 A load or store of an atomic object should be implemented by a single load or store instruction. See C.6(23/2).

103/2 If the target domain requires deterministic memory use at run time, storage for task attributes should be pre-allocated statically and the number of attributes pre-allocated should be documented. See C.7.2(30).

104/2 Finalization of task attributes and reclamation of associated storage should be performed as soon as possible after task termination. See C.7.2(30.1/2).

105/2 Names that end with "_Locking" should be used for implementation-defined locking policies. See D.3(17).

106/2 Names that end with "_Queuing" should be used for implementation-defined queuing policies. See D.4(16).

107/2 The abort_statement should not require the task executing the statement to block. See D.6(9).

108/2 On a multi-processor, the delay associated with aborting a task on another processor should be bounded. See D.6(10).

109/2 When feasible, specified restrictions should be used to produce a more efficient implementation. See D.7(21).

110/2 When appropriate, mechanisms to change the value of Tick should be provided. See D.8(47).

111/2 Calendar.Clock and Real_Time.Clock should be transformations of the same time base. See D.8(48).

112/2 The “best” time base which exists in the underlying system should be available to the application through Real_Time.Clock. See D.8(49).

112.1/3 On a multiprocessor system, each processor should have a separate and disjoint ready queue. See D.13(9).

113/2 When appropriate, implementations should provide configuration mechanisms to change the value of Execution_Time.CPU_Tick. See D.14(29/2).

114/2 For a timing event, the handler should be executed directly by the real-time clock interrupt mechanism. See D.15(25).

114.1/3 Each dispatching domain should have separate and disjoint ready queues. See D.16.1(31).

115/2 The PCS should allow for multiple tasks to call the RPC-receiver. See E.5(28).

116/2 The System.RPC.Write operation should raise Storage_Error if it runs out of space when writing an item. See E.5(29).

117/2 If COBOL (respectively, C) is supported in the target environment, then interfacing to COBOL (respectively, C) should be supported as specified in Annex B. See F(7/3).

118/2 Packed decimal should be used as the internal representation for objects of subtype S when S'Machine_Radix = 10. See F.1(2).

119/2 If Fortran (respectively, C) is supported in the target environment, then interfacing to Fortran (respectively, C) should be supported as specified in Annex B. See G(7/3).

120/2 Mixed real and complex operations (as well as pure-imaginary and complex operations) should not be performed by converting the real (resp. pure-imaginary) operand to complex. See G.1.1(56).

121/3 If Real'Signed_Zeros is True for Numerics.Generic_Complex_Types, a rational treatment of the signs of zero results and result components should be provided. See G.1.1(58).

122/3 If Complex_Types.Real'Signed_Zeros is True for Numerics.Generic_Complex_Elementary_Functions, a rational treatment of the signs of zero results and result components should be provided. See G.1.2(49).

123/2 For elementary functions, the forward trigonometric functions without a Cycle parameter should not be implemented by calling the corresponding version with a Cycle parameter. Log without a Base parameter should not be implemented by calling Log with a Base parameter. See G.2.4(19).

124/2 For complex arithmetic, the Compose_From_Polar function without a Cycle parameter should not be implemented by calling Compose_From_Polar with a Cycle parameter. See G.2.6(15).

125/2 Solve and Inverse for Numerics.Generic_Real_Arrays should be implemented using

established techniques such as LU decomposition and the result should be refined by an iteration on the residuals. See G.3.1(88/3).

126/2 The equality operator should be used to test that a matrix in Numerics.Generic_Real_Arrays is symmetric. See G.3.1(90/2).

126.1/3 An implementation should minimize the circumstances under which the algorithm used for Numerics.Generic_Real_Arrays.Eigenvalues and Numerics.Generic_Real_Arrays.Eigensystem fails to converge. See G.3.1(91/3).

127/2 Solve and Inverse for Numerics.Generic_Complex_Arrays should be implemented using established techniques and the result should be refined by an iteration on the residuals. See G.3.2(158/3).

128/2 The equality and negation operators should be used to test that a matrix is Hermitian. See G.3.2(160/2).

128.1/3 An implementation should minimize the circumstances under which the algorithm used for Numerics.Generic_Complex_Arrays.Eigenvalues and Numerics.Generic_Complex_Arrays.Eigensystem fails to converge. See G.3.2(160.1/3).

129/2 Mixed real and complex operations should not be performed by converting the real operand to complex. See G.3.2(161/2).

130/2 The information produced by pragma Reviewable should be provided in both a human-readable and machine-readable form, and the latter form should be documented. See H.3.1(19).

131/2 Object code listings should be provided both in a symbolic format and in a numeric format. See H.3.1(20).

132/3 If the partition elaboration policy is Sequential and the Environment task becomes permanently blocked during elaboration, then the partition should be immediately terminated. See H.6(15/3).