Foreword to this version of the Ada Reference Manual

The International Standard for the programming language Ada is ISO/IEC 8652:1995(E). 0.1/1

The Ada Working Group ISO/IEC JTC1/SC 22/WG 9 is tasked by ISO with the work item to interpret and maintain the International Standard and to produce Technical Corrigenda, as appropriate. The technical work on the International Standard is performed by the Ada Rapporteur Group (ARG) of WG 9. In September 2000, WG 9 approved and forwarded Technical Corrigendum 1 to SC 22 for ISO approval, which was granted in February 2001. 0.2/1

The Technical Corrigendum lists the individual changes that need to be made to the text of the International Standard to correct errors, omissions or inconsistencies. Once approved, the corrections specified in Technical Corrigendum 1 will be part of the International Standard ISO/IEC 8652:1995(E). 0.3/1

When ISO publishes Technical Corrigendum 1, it is unlikely that ISO will also publish a document that merges the Technical Corrigendum changes into the text of the International Standard. However, ISO rules require that the project editor for the Technical Corrigendum be able to produce such a document on demand. 0.4/1

This version of the Ada Reference Manual is what the project editor would provide to ISO in response to such a request. It incorporates the changes specified in the Technical Corrigendum into the text of ISO/IEC 8652:1995(E). It should be understood that the publication of any ISO document involves changes in general format, boilerplate, headers, etc., as well as a review by professional editors that may introduce editorial changes to the text. This version of the Ada Reference Manual is therefore neither an official ISO document, nor a version guaranteed to be identical to an official ISO document, should ISO decide to reprint the International Standard incorporating an approved Technical Corrigendum. It is nevertheless a best effort to be as close as possible to the technical content of such an updated document. In the case of a conflict between this document and a Technical Corrigendum 1 approved by ISO (or between this document and the original 8652:1995 in the case of paragraphs not changed by Technical Corrigendum 1), the other documents contain the official text of the International Standard ISO/IEC 8652:1995(E). 0.5/1

As it is very inconvenient to have the Reference Manual for Ada specified in two documents, this consolidated version of the Ada Reference Manual is made available to the public. 0.6/1

Using this version of the Ada Reference Manual

This document has been revised with the corrections specified in Technical Corrigendum 1 (ISO/IEC 8652:1995/COR1:2000). In addition, a variety of editorial errors have been corrected. 0.7/1

Changes to the original 8652:1995 can be identified by the version number /1 following the paragraph number. Paragraphs not so marked are unchanged by Technical Corrigendum 1 or editorial corrections. Paragraph numbers of unchanged paragraphs are the same as in the original Ada Reference Manual. In addition, some versions of this document include revision bars near the paragraph numbers. Where paragraphs are inserted, the paragraph numbers are of the form pp.nn, where pp is the number of the preceding paragraph, and nn is an insertion number. For instance, the first paragraph inserted after paragraph 8 is numbered 8.1, the second paragraph inserted is numbered 8.2, and so on. Deleted paragraphs are indicated by the text *This paragraph was deleted.* Deleted paragraphs include empty paragraphs that were numbered in the original Ada Reference Manual. 0.8/1

Acknowledgements for this version of the Ada Reference Manual

0.9/1 The editor [R. Brukardt (USA)] would like to thank the many people whose hard work and assistance has made this revision possible.

0.10/1 Thanks go out to all of the members of the ISO/IEC JTC 1/SC 22/WG 9 Ada Rapporteur Group, whose work on creating and editing the wording corrections was critical to the entire process. Especially valuable contributions came from the chairman of the ARG, E. Ploedereder (Germany), who kept the process moving; J. Barnes (UK) and K. Ishihata (Japan), whose extremely detailed reviews kept the editor on his toes; G. Dismukes (USA), M. Kamrad (USA), P. Leroy (France), S. Michell (Canada), T. Taft (USA), J. Tokar (USA), and other members too numerous to mention.

0.11/1 Special thanks go to R. Duff (USA) for his explanations of the previous system of formatting of these documents during the tedious conversion to more modern formats. Special thanks also go to the convener of ISO/IEC JTC 1/SC 22/WG 9, J. Moore (USA), without whose help and support the corrigendum and this consolidated reference manual would not have been possible.

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. 1

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote. 2

International Standard ISO/IEC 8652 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*. 3

This second edition cancels and replaces the first edition (ISO 8652:1987), of which it constitutes a technical revision. 4

Annexes A to J form an integral part of this International Standard. Annexes K to P are for information only. 5

Introduction

1 This is the Ada Reference Manual.

2 Other available Ada documents include:

3 • Rationale for the Ada Programming Language — 1995 edition, which gives an introduction to the new features of Ada, and explains the rationale behind them. Programmers should read this first.

4/1 • *This paragraph was deleted.*

5 • The Annotated Ada Reference Manual (AARM). The AARM contains all of the text in the RM95, plus various annotations. It is intended primarily for compiler writers, validation test writers, and others who wish to study the fine details. The annotations include detailed rationale for individual rules and explanations of some of the more arcane interactions among the rules.

Design Goals

6 Ada was originally designed with three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency. This revision to the language was designed to provide greater flexibility and extensibility, additional control over storage management and synchronization, and standardized packages oriented toward supporting important application areas, while at the same time retaining the original emphasis on reliability, maintainability, and efficiency.

7 The need for languages that promote reliability and simplify maintenance is well established. Hence emphasis was placed on program readability over ease of writing. For example, the rules of the language require that program variables be explicitly declared and that their type be specified. Since the type of a variable is invariant, compilers can ensure that operations on variables are compatible with the properties intended for objects of the type. Furthermore, error-prone notations have been avoided, and the syntax of the language avoids the use of encoded forms in favor of more English-like constructs. Finally, the language offers support for separate compilation of program units in a way that facilitates program development and maintenance, and which provides the same degree of checking between units as within a unit.

8 Concern for the human programmer was also stressed during the design. Above all, an attempt was made to keep to a relatively small number of underlying concepts integrated in a consistent and systematic way while continuing to avoid the pitfalls of excessive involution. The design especially aims to provide language constructs that correspond intuitively to the normal expectations of users.

9 Like many other human activities, the development of programs is becoming ever more decentralized and distributed. Consequently, the ability to assemble a program from independently produced software components continues to be a central idea in the design. The concepts of packages, of private types, and of generic units are directly related to this idea, which has ramifications in many other aspects of the language. An allied concern is the maintenance of programs to match changing requirements; type extension and the hierarchical library enable a program to be modified while minimizing disturbance to existing tested and trusted components.

10 No language can avoid the problem of efficiency. Languages that require over-elaborate compilers, or that lead to the inefficient use of storage or execution time, force these inefficiencies on all machines and on all programs. Every construct of the language was examined in the light of present implementation techniques. Any proposed construct whose implementation was unclear or that required excessive machine resources was rejected.

Language Summary

An Ada program is composed of one or more program units. Program units may be subprograms (which define executable algorithms), packages (which define collections of entities), task units (which define concurrent computations), protected units (which define operations for the coordinated sharing of data between tasks), or generic units (which define parameterized forms of packages and subprograms). Each program unit normally consists of two parts: a specification, containing the information that must be visible to other units, and a body, containing the implementation details, which need not be visible to other units. Most program units can be compiled separately. 11

This distinction of the specification and body, and the ability to compile units separately, allows a program to be designed, written, and tested as a set of largely independent software components. 12

An Ada program will normally make use of a library of program units of general utility. The language provides means whereby individual organizations can construct their own libraries. All libraries are structured in a hierarchical manner; this enables the logical decomposition of a subsystem into individual components. The text of a separately compiled program unit must name the library units it requires. 13

Program Units 14

A subprogram is the basic unit for expressing an algorithm. There are two kinds of subprograms: procedures and functions. A procedure is the means of invoking a series of actions. For example, it may read data, update variables, or produce some output. It may have parameters, to provide a controlled means of passing information between the procedure and the point of call. A function is the means of invoking the computation of a value. It is similar to a procedure, but in addition will return a result. 15

A package is the basic unit for defining a collection of logically related entities. For example, a package can be used to define a set of type declarations and associated operations. Portions of a package can be hidden from the user, thus allowing access only to the logical properties expressed by the package specification. 16

Subprogram and package units may be compiled separately and arranged in hierarchies of parent and child units giving fine control over visibility of the logical properties and their detailed implementation. 17

A task unit is the basic unit for defining a task whose sequence of actions may be executed concurrently with those of other tasks. Such tasks may be implemented on multicomputers, multiprocessors, or with interleaved execution on a single processor. A task unit may define either a single executing task or a task type permitting the creation of any number of similar tasks. 18

A protected unit is the basic unit for defining protected operations for the coordinated use of data shared between tasks. Simple mutual exclusion is provided automatically, and more elaborate sharing protocols can be defined. A protected operation can either be a subprogram or an entry. A protected entry specifies a Boolean expression (an entry barrier) that must be true before the body of the entry is executed. A protected unit may define a single protected object or a protected type permitting the creation of several similar objects. 19

Declarations and Statements 20

The body of a program unit generally contains two parts: a declarative part, which defines the logical entities to be used in the program unit, and a sequence of statements, which defines the execution of the program unit. 21

22 The declarative part associates names with declared entities. For example, a name may denote a type, a constant, a variable, or an exception. A declarative part also introduces the names and parameters of other nested subprograms, packages, task units, protected units, and generic units to be used in the program unit.

23 The sequence of statements describes a sequence of actions that are to be performed. The statements are executed in succession (unless a transfer of control causes execution to continue from another place).

24 An assignment statement changes the value of a variable. A procedure call invokes execution of a procedure after associating any actual parameters provided at the call with the corresponding formal parameters.

25 Case statements and if statements allow the selection of an enclosed sequence of statements based on the value of an expression or on the value of a condition.

26 The loop statement provides the basic iterative mechanism in the language. A loop statement specifies that a sequence of statements is to be executed repeatedly as directed by an iteration scheme, or until an exit statement is encountered.

27 A block statement comprises a sequence of statements preceded by the declaration of local entities used by the statements.

28 Certain statements are associated with concurrent execution. A delay statement delays the execution of a task for a specified duration or until a specified time. An entry call statement is written as a procedure call statement; it requests an operation on a task or on a protected object, blocking the caller until the operation can be performed. A called task may accept an entry call by executing a corresponding accept statement, which specifies the actions then to be performed as part of the rendezvous with the calling task. An entry call on a protected object is processed when the corresponding entry barrier evaluates to true, whereupon the body of the entry is executed. The requeue statement permits the provision of a service as a number of related activities with preference control. One form of the select statement allows a selective wait for one of several alternative rendezvous. Other forms of the select statement allow conditional or timed entry calls and the asynchronous transfer of control in response to some triggering event.

29 Execution of a program unit may encounter error situations in which normal program execution cannot continue. For example, an arithmetic computation may exceed the maximum allowed value of a number, or an attempt may be made to access an array component by using an incorrect index value. To deal with such error situations, the statements of a program unit can be textually followed by exception handlers that specify the actions to be taken when the error situation arises. Exceptions can be raised explicitly by a raise statement.

30 *Data Types*

31 Every object in the language has a type, which characterizes a set of values and a set of applicable operations. The main classes of types are elementary types (comprising enumeration, numeric, and access types) and composite types (including array and record types).

32 An enumeration type defines an ordered set of distinct enumeration literals, for example a list of states or an alphabet of characters. The enumeration types Boolean, Character, and Wide_Character are predefined.

33 Numeric types provide a means of performing exact or approximate numerical computations. Exact computations use integer types, which denote sets of consecutive integers. Approximate computations use either fixed point types, with absolute bounds on the error, or floating point types, with relative bounds on the error. The numeric types Integer, Float, and Duration are predefined.

Composite types allow definitions of structured objects with related components. The composite types in the language include arrays and records. An array is an object with indexed components of the same type. A record is an object with named components of possibly different types. Task and protected types are also forms of composite types. The array types String and Wide_String are predefined. 34

Record, task, and protected types may have special components called discriminants which parameterize the type. Variant record structures that depend on the values of discriminants can be defined within a record type. 35

Access types allow the construction of linked data structures. A value of an access type represents a reference to an object declared as aliased or to an object created by the evaluation of an allocator. Several variables of an access type may designate the same object, and components of one object may designate the same or other objects. Both the elements in such linked data structures and their relation to other elements can be altered during program execution. Access types also permit references to subprograms to be stored, passed as parameters, and ultimately dereferenced as part of an indirect call. 36

Private types permit restricted views of a type. A private type can be defined in a package so that only the logically necessary properties are made visible to the users of the type. The full structural details that are externally irrelevant are then only available within the package and any child units. 37

From any type a new type may be defined by derivation. A type, together with its derivatives (both direct and indirect) form a derivation class. Class-wide operations may be defined that accept as a parameter an operand of any type in a derivation class. For record and private types, the derivatives may be extensions of the parent type. Types that support these object-oriented capabilities of class-wide operations and type extension must be tagged, so that the specific type of an operand within a derivation class can be identified at run time. When an operation of a tagged type is applied to an operand whose specific type is not known until run time, implicit dispatching is performed based on the tag of the operand. 38

The concept of a type is further refined by the concept of a subtype, whereby a user can constrain the set of allowed values of a type. Subtypes can be used to define subranges of scalar types, arrays with a limited set of index values, and records and private types with particular discriminant values. 39

Other Facilities 40

Representation clauses can be used to specify the mapping between types and features of an underlying machine. For example, the user can specify that objects of a given type must be represented with a given number of bits, or that the components of a record are to be represented using a given storage layout. Other features allow the controlled use of low level, nonportable, or implementation-dependent aspects, including the direct insertion of machine code. 41

The predefined environment of the language provides for input-output and other capabilities (such as string manipulation and random number generation) by means of standard library packages. Input-output is supported for values of user-defined as well as of predefined types. Standard means of representing values in display form are also provided. Other standard library packages are defined in annexes of the standard to support systems with specialized requirements. 42

Finally, the language provides a powerful means of parameterization of program units, called generic program units. The generic parameters can be types and subprograms (as well as objects and packages) and so allow general algorithms and data structures to be defined that are applicable to all types of a given class. 43

Language Changes

44 This International Standard replaces the first edition of 1987. In this edition, the following major language changes have been incorporated:

- 45 Support for standard 8-bit and 16-bit character sets. See Section 2, 3.5.2, 3.6.3, A.1, A.3, and A.4.
- 46 Object-oriented programming with run-time polymorphism. See the discussions of classes, derived types, tagged types, record extensions, and private extensions in clauses 3.4, 3.9, and 7.3. See also the new forms of generic formal parameters that are allowed by 12.5.1, ‘‘Formal Private and Derived Types’’ and 12.7, ‘‘Formal Packages’’.
- 47 Access types have been extended to allow an access value to designate a subprogram or an object declared by an object declaration (as opposed to just a heap-allocated object). See 3.10.
- 48 Efficient data-oriented synchronization is provided via protected types. See Section 9.
- 49 The library units of a library may be organized into a hierarchy of parent and child units. See Section 10.
- 50 Additional support has been added for interfacing to other languages. See Annex B.
- 51 The Specialized Needs Annexes have been added to provide specific support for certain application areas:
  - 52 Annex C, ‘‘Systems Programming’’
  - 53 Annex D, ‘‘Real-Time Systems’’
  - 54 Annex E, ‘‘Distributed Systems’’
  - 55 Annex F, ‘‘Information Systems’’
  - 56 Annex G, ‘‘Numerics’’
  - 57 Annex H, ‘‘Safety and Security’’

Instructions for Comment Submission

Informal comments on this International Standard may be sent via e-mail to **ada-comment@ada-auth.org**. If appropriate, the Project Editor will initiate the defect correction procedure. 58/1

Comments should use the following format: 59

!topic *Title summarizing comment* 60
!reference RM95-*ss.ss(pp)*
!from *Author Name yy-mm-dd*
!keywords *keywords related to topic*
!discussion

text of discussion

where *ss.ss* is the section, clause or subclause number, *pp* is the paragraph number where applicable, and *yy-mm-dd* is the date the comment was sent. The date is optional, as is the **!keywords** line. 61

Please use a descriptive ‘‘Subject’’ in your e-mail message, and limit each message to a single comment. 62/1

When correcting typographical errors or making minor wording suggestions, please put the correction directly as the topic of the comment; use square brackets [ ] to indicate text to be omitted and curly braces { } to indicate text to be added, and provide enough context to make the nature of the suggestion self-evident or put additional information in the body of the comment, for example: 63

!topic [c]{C}haracter 64
!topic it[']s meaning is not defined

Formal requests for interpretations and for reporting defects in this International Standard may be made in accordance with the ISO/IEC JTC1 Directives and the ISO/IEC JTC1/SC22 policy for interpretations. National Bodies may submit a Defect Report to ISO/IEC JTC1/SC22 for resolution under the JTC1 procedures. A response will be provided and, if appropriate, a Technical Corrigendum will be issued in accordance with the procedures. 65

Acknowledgements

66 This International Standard was prepared by the Ada 9X Mapping/Revision Team based at Intermetrics, Inc., which has included: W. Carlson, Program Manager; T. Taft, Technical Director; J. Barnes (consultant); B. Brosgol (consultant); R. Duff (Oak Tree Software); M. Edwards; C. Garrity; R. Hilliard; O. Pazy (consultant); D. Rosenfeld; L. Shafer; W. White; M. Woodger.

67 The following consultants to the Ada 9X Project contributed to the Specialized Needs Annexes: T. Baker (Real-Time/Systems Programming — SEI, FSU); K. Dritz (Numerics — Argonne National Laboratory); A. Gargaro (Distributed Systems — Computer Sciences); J. Goodenough (Real-Time/Systems Programming — SEI); J. McHugh (Secure Systems — consultant); B. Wichmann (Safety-Critical Systems — NPL: UK).

68 This work was regularly reviewed by the Ada 9X Distinguished Reviewers and the members of the Ada 9X Rapporteur Group (XRG): E. Ploedereder, Chairman of DRs and XRG (University of Stuttgart: Germany); B. Bardin (Hughes); J. Barnes (consultant: UK); B. Brett (DEC); B. Brosgol (consultant); R. Brukardt (RR Software); N. Cohen (IBM); R. Dewar (NYU); G. Dismukes (TeleSoft); A. Evans (consultant); A. Gargaro (Computer Sciences); M. Gerhardt (ESL); J. Goodenough (SEI); S. Heilbrunner (University of Salzburg: Austria); P. Hilfinger (UC/Berkeley); B. Källberg (CelsiusTech: Sweden); M. Kamrad II (Unisys); J. van Katwijk (Delft University of Technology: The Netherlands); V. Kaufman (Russia); P. Kruchten (Rational); R. Landwehr (CCI: Germany); C. Lester (Portsmouth Polytechnic: UK); L. Månsson (TELIA Research: Sweden); S. Michell (Multiprocessor Toolsmiths: Canada); M. Mills (US Air Force); D. Pogge (US Navy); K. Power (Boeing); O. Roubine (Verdix: France); A. Strohmeier (Swiss Fed Inst of Technology: Switzerland); W. Taylor (consultant: UK); J. Tokar (Tartan); E. Vasilescu (Grumman); J. Vladik (Prospeks s.r.o.: Czech Republic); S. Van Vlierberghe (OFFIS: Belgium).

69 Other valuable feedback influencing the revision process was provided by the Ada 9X Language Precision Team (Odyssey Research Associates), the Ada 9X User/Implementer Teams (AETECH, Tartan, TeleSoft), the Ada 9X Implementation Analysis Team (New York University) and the Ada community-at-large.

70 Special thanks go to R. Mathis, Convenor of ISO/IEC JTC1/SC22 Working Group 9.

71 The Ada 9X Project was sponsored by the Ada Joint Program Office. Christine M. Anderson at the Air Force Phillips Laboratory (Kirtland AFB, NM) was the project manager.

Changes

The International Standard is the same as this version of the Reference Manual, except: 72

- This list of Changes is not included in the International Standard. 73
- The ‘‘Acknowledgements’’ page is not included in the International Standard. 74
- The text in the running headers and footers on each page is slightly different in the International Standard. 75
- The title page(s) are different in the International Standard. 76
- This document is formatted for 8.5-by-11-inch paper, whereas the International Standard is formatted for A4 paper (210-by-297mm); thus, the page breaks are in different places. 77
- The ‘‘Foreword to this version of the Ada Reference Manual’’ clause is not included in the International Standard. 77.1/1