INTERNATIONAL STANDARD

ISO/IEC 8652

Fourth edition
2023-05

Information technology — Programming languages — Ada

Technologies de l'information — Langages de programmation — Ada

Reference number
ISO/IEC 8652:2023(E)

© ISO/IEC 2023



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2023

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO/IEC 2023 – All rights reserved

Contents

Foreword ... xiii

Introduction ... xv

1 Scope ... 1

2 Normative references ... 1

3 Terms and definitions ... 2

3.1 Types, objects, and their properties ... 2
3.2 Subprograms and their properties ... 7
3.3 Other syntactic constructs ... 7
3.4 Runtime actions ... 10
3.5 Exceptional situations ... 10

4 General ... 11

4.1 Structure ... 11
4.2 Conformity of an Implementation ... 12
4.3 Method of Description and Syntax Notation ... 14
4.4 Classification of Errors ... 15

5 Lexical Elements ... 16

5.1 Character Set ... 16
5.2 Lexical Elements, Separators, and Delimiters ... 18
5.3 Identifiers ... 19
5.4 Numeric Literals ... 20
5.4.1 Decimal Literals ... 20
5.4.2 Based Literals ... 21
5.5 Character Literals ... 21
5.6 String Literals ... 21
5.7 Comments ... 22
5.8 Pragmas ... 22
5.9 Reserved Words ... 25

6 Declarations and Types ... 25

6.1 Declarations ... 25
6.2 Types and Subtypes ... 27
6.2.1 Type Declarations ... 28
6.2.2 Subtype Declarations ... 29
6.2.3 Classification of Operations ... 30
6.2.4 Subtype Predicates ... 31
6.3 Objects and Named Numbers ... 34
6.3.1 Object Declarations ... 36
6.3.2 Number Declarations ... 39
6.4 Derived Types and Classes ... 39
6.4.1 Derivation Classes ... 42
6.5 Scalar Types ... 44
6.5.1 Enumeration Types ... 47
6.5.2 Character Types ... 48
6.5.3 Boolean Types ... 49
6.5.4 Integer Types ... 49
6.5.5 Operations of Discrete Types ... 52
6.5.6 Real Types ... 53
6.5.7 Floating Point Types ... 54

© ISO/IEC 2023 – All rights reserved

6.5.8 Operations of Floating Point Types ... 55
6.5.9 Fixed Point Types ... 56
6.5.10 Operations of Fixed Point Types ... 58
6.6 Array Types ... 59
6.6.1 Index Constraints and Discrete Ranges ... 61
6.6.2 Operations of Array Types ... 62
6.6.3 String Types ... 63
6.7 Discriminants ... 63
6.7.1 Discriminant Constraints ... 66
6.7.2 Operations of Discriminated Types ... 67
6.8 Record Types ... 67
6.8.1 Variant Parts and Discrete Choices ... 70
6.9 Tagged Types and Type Extensions ... 72
6.9.1 Type Extensions ... 75
6.9.2 Dispatching Operations of Tagged Types ... 76
6.9.3 Abstract Types and Subprograms ... 79
6.9.4 Interface Types ... 80
6.10 Access Types ... 82
6.10.1 Incomplete Type Declarations ... 85
6.10.2 Operations of Access Types ... 87
6.11 Declarative Parts ... 93
6.11.1 Completions of Declarations ... 93
7 Names and Expressions ... 94
7.1 Names ... 94
7.1.1 Indexed Components ... 95
7.1.2 Slices ... 96
7.1.3 Selected Components ... 97
7.1.4 Attributes ... 99
7.1.5 User-Defined References ... 100
7.1.6 User-Defined Indexing ... 101
7.2 Literals ... 103
7.2.1 User-Defined Literals ... 104
7.3 Aggregates ... 106
7.3.1 Record Aggregates ... 106
7.3.2 Extension Aggregates ... 108
7.3.3 Array Aggregates ... 110
7.3.4 Delta Aggregates ... 114
7.3.5 Container Aggregates ... 115
7.4 Expressions ... 121
7.5 Operators and Expression Evaluation ... 123
7.5.1 Logical Operators and Short-circuit Control Forms ... 124
7.5.2 Relational Operators and Membership Tests ... 125
7.5.3 Binary Adding Operators ... 128
7.5.4 Unary Adding Operators ... 129
7.5.5 Multiplying Operators ... 129
7.5.6 Highest Precedence Operators ... 132
7.5.7 Conditional Expressions ... 132
7.5.8 Quantified Expressions ... 134
7.5.9 Declare Expressions ... 135
7.5.10 Reduction Expressions ... 136
7.6 Type Conversions ... 139
7.7 Qualified Expressions ... 144
7.8 Allocators ... 144
7.9 Static Expressions and Static Subtypes ... 146
7.9.1 Statically Matching Constraints and Subtypes ... 150

 © ISO/IEC 2023 – All rights reserved

7.10 Image Attributes ... 151
8 Statements ... 154
8.1 Simple and Compound Statements - Sequences of Statements ... 155
8.2 Assignment Statements ... 156
8.2.1 Target Name Symbols ... 158
8.3 If Statements ... 158
8.4 Case Statements ... 159
8.5 Loop Statements ... 160
8.5.1 User-Defined Iterator Types ... 164
8.5.2 Generalized Loop Iteration ... 166
8.5.3 Procedural Iterators ... 169
8.6 Block Statements ... 172
8.6.1 Parallel Block Statements ... 172
8.7 Exit Statements ... 174
8.8 Goto Statements ... 175
9 Subprograms ... 175
9.1 Subprogram Declarations ... 176
9.1.1 Preconditions and Postconditions ... 178
9.1.2 The Global and Global'Class Aspects ... 183
9.2 Formal Parameter Modes ... 186
9.3 Subprogram Bodies ... 187
9.3.1 Conformance Rules ... 188
9.3.2 Inline Expansion of Subprograms ... 190
9.4 Subprogram Calls ... 190
9.4.1 Parameter Associations ... 192
9.5 Return Statements ... 194
9.5.1 Nonreturning Subprograms ... 197
9.6 Overloading of Operators ... 198
9.7 Null Procedures ... 199
9.8 Expression Functions ... 199
10 Packages ... 201
10.1 Package Specifications and Declarations ... 201
10.2 Package Bodies ... 202
10.3 Private Types and Private Extensions ... 203
10.3.1 Private Operations ... 205
10.3.2 Type Invariants ... 208
10.3.3 Default Initial Conditions ... 211
10.3.4 Stable Properties of a Type ... 211
10.4 Deferred Constants ... 213
10.5 Limited Types ... 214
10.6 Assignment and Finalization ... 216
10.6.1 Completion and Finalization ... 219
11 Visibility Rules ... 221
11.1 Declarative Region ... 221
11.2 Scope of Declarations ... 222
11.3 Visibility ... 223
11.3.1 Overriding Indicators ... 226
11.4 Use Clauses ... 227
11.5 Renaming Declarations ... 228
11.5.1 Object Renaming Declarations ... 228
11.5.2 Exception Renaming Declarations ... 230
11.5.3 Package Renaming Declarations ... 230
11.5.4 Subprogram Renaming Declarations ... 230

© ISO/IEC 2023 – All rights reserved

11.5.5 Generic Renaming Declarations 232
11.6 The Context of Overload Resolution 233
12 Tasks and Synchronization 235
12.1 Task Units and Task Objects 236
12.2 Task Execution - Task Activation 238
12.3 Task Dependence - Termination of Tasks 239
12.4 Protected Units and Protected Objects 241
12.5 Intertask Communication 244
12.5.1 Protected Subprograms and Protected Actions 248
12.5.2 Entries and Accept Statements 250
12.5.3 Entry Calls 253
12.5.4 Requeue Statements 255
12.6 Delay Statements, Duration, and Time 257
12.6.1 Formatting, Time Zones, and other operations for Time 260
12.7 Select Statements 266
12.7.1 Selective Accept 266
12.7.2 Timed Entry Calls 268
12.7.3 Conditional Entry Calls 269
12.7.4 Asynchronous Transfer of Control 269
12.8 Abort of a Task - Abort of a Sequence of Statements 270
12.9 Task and Entry Attributes 272
12.10 Shared Variables 272
12.10.1 Conflict Check Policies 274
12.11 Example of Tasking and Synchronization 276
13 Program Structure and Compilation Issues 277
13.1 Separate Compilation 277
13.1.1 Compilation Units - Library Units 278
13.1.2 Context Clauses - With Clauses 281
13.1.3 Subunits of Compilation Units 283
13.1.4 The Compilation Process 284
13.1.5 Pragmas and Program Units 285
13.1.6 Environment-Level Visibility Rules 286
13.2 Program Execution 286
13.2.1 Elaboration Control 288
14 Exceptions 291
14.1 Exception Declarations 292
14.2 Exception Handlers 292
14.3 Raise Statements and Raise Expressions 293
14.4 Exception Handling 294
14.4.1 The Package Exceptions 295
14.4.2 Pragmas Assert and Assertion_Policy 297
14.4.3 Example of Exception Handling 299
14.5 Suppressing Checks 300
14.6 Exceptions and Optimization 303
15 Generic Units 304
15.1 Generic Declarations 304
15.2 Generic Bodies 306
15.3 Generic Instantiation 307
15.4 Formal Objects 309
15.5 Formal Types 310
15.5.1 Formal Private and Derived Types 312
15.5.2 Formal Scalar Types 314
15.5.3 Formal Array Types 314

 © ISO/IEC 2023 – All rights reserved

15.5.4 Formal Access Types ... 315
15.5.5 Formal Interface Types ... 316
15.6 Formal Subprograms ... 316
15.7 Formal Packages ... 319
15.8 Example of a Generic Package ... 321
16 Representation Issues ... 322
16.1 Operational and Representation Aspects ... 322
16.1.1 Aspect Specifications ... 326
16.2 Packed Types ... 329
16.3 Operational and Representation Attributes ... 330
16.4 Enumeration Representation Clauses ... 337
16.5 Record Layout ... 338
16.5.1 Record Representation Clauses ... 338
16.5.2 Storage Place Attributes ... 341
16.5.3 Bit Ordering ... 341
16.6 Change of Representation ... 342
16.7 The Package System ... 342
16.7.1 The Package System.Storage_Elements ... 344
16.7.2 The Package System.Address_To_Access_Conversions ... 345
16.8 Machine Code Insertions ... 346
16.9 Unchecked Type Conversions ... 347
16.9.1 Data Validity ... 348
16.9.2 The Valid Attribute ... 349
16.10 Unchecked Access Value Creation ... 349
16.11 Storage Management ... 350
16.11.1 Storage Allocation Attributes ... 353
16.11.2 Unchecked Storage Deallocation ... 354
16.11.3 Default Storage Pools ... 355
16.11.4 Storage Subpools ... 356
16.11.5 Subpool Reclamation ... 359
16.11.6 Storage Subpool Example ... 359
16.12 Pragma Restrictions and Pragma Profile ... 362
16.12.1 Language-Defined Restrictions and Profiles ... 363
16.13 Streams ... 365
16.13.1 The Streams Subsystem ... 365
16.13.2 Stream-Oriented Attributes ... 368
16.14 Freezing Rules ... 373
Annex A (normative) Predefined Language Environment ... 376
A.1 The Package Standard ... 379
A.2 The Package Ada ... 384
A.3 Character Handling ... 384
A.3.1 The Packages Characters, Wide_Characters, and Wide_Wide_Characters ... 384
A.3.2 The Package Characters.Handling ... 385
A.3.3 The Package Characters.Latin_1 ... 387
A.3.4 The Package Characters.Conversions ... 395
A.3.5 The Package Wide_Characters.Handling ... 397
A.3.6 The Package Wide_Wide_Characters.Handling ... 400
A.4 String Handling ... 400
A.4.1 The Package Strings ... 400
A.4.2 The Package Strings.Maps ... 401
A.4.3 Fixed-Length String Handling ... 404
A.4.4 Bounded-Length String Handling ... 412
A.4.5 Unbounded-Length String Handling ... 419
A.4.6 String-Handling Sets and Mappings ... 425

© ISO/IEC 2023 – All rights reserved

A.4.7 Wide_String Handling 425
A.4.8 Wide_Wide_String Handling 428
A.4.9 String Hashing 430
A.4.10 String Comparison 431
A.4.11 String Encoding 432
A.4.12 Universal Text Buffers 437
A.5 The Numerics Packages 439
A.5.1 Elementary Functions 440
A.5.2 Random Number Generation 443
A.5.3 Attributes of Floating Point Types 448
A.5.4 Attributes of Fixed Point Types 452
A.5.5 Big Numbers 453
A.5.6 Big Integers 453
A.5.7 Big Reals 455
A.6 Input-Output 457
A.7 External Files and File Objects 457
A.8 Sequential and Direct Files 458
A.8.1 The Generic Package Sequential_IO 459
A.8.2 File Management 460
A.8.3 Sequential Input-Output Operations 462
A.8.4 The Generic Package Direct_IO 463
A.8.5 Direct Input-Output Operations 464
A.9 The Generic Package Storage_IO 465
A.10 Text Input-Output 466
A.10.1 The Package Text_IO 467
A.10.2 Text File Management 475
A.10.3 Default Input, Output, and Error Files 475
A.10.4 Specification of Line and Page Lengths 476
A.10.5 Operations on Columns, Lines, and Pages 477
A.10.6 Get and Put Procedures 480
A.10.7 Input-Output of Characters and Strings 481
A.10.8 Input-Output for Integer Types 483
A.10.9 Input-Output for Real Types 485
A.10.10 Input-Output for Enumeration Types 487
A.10.11 Input-Output for Bounded Strings 488
A.10.12 Input-Output for Unbounded Strings 489
A.11 Wide Text Input-Output and Wide Wide Text Input-Output 491
A.12 Stream Input-Output 491
A.12.1 The Package Streams.Stream_IO 491
A.12.2 The Package Text_IO.Text_Streams 494
A.12.3 The Package Wide_Text_IO.Text_Streams 495
A.12.4 The Package Wide_Wide_Text_IO.Text_Streams 495
A.13 Exceptions in Input-Output 495
A.14 File Sharing 497
A.15 The Package Command_Line 497
A.15.1 The Packages Wide_Command_Line and Wide_Wide_Command_Line 498
A.16 The Package Directories 498
A.16.1 The Package Directories.Hierarchical_File_Names 506
A.16.2 The Packages Wide_Directories and Wide_Wide_Directories 507
A.17 The Package Environment_Variables 507
A.17.1 The Packages Wide_Environment_Variables and Wide_Wide_Environment_Variables 509
A.18 Containers 509
A.18.1 The Package Containers 511
A.18.2 The Generic Package Containers.Vectors 511

 © ISO/IEC 2023 – All rights reserved

A.18.3 The Generic Package Containers.Doubly_Linked_Lists ... 545
A.18.4 Maps ... 567
A.18.5 The Generic Package Containers.Hashed_Maps ... 577
A.18.6 The Generic Package Containers.Ordered_Maps ... 586
A.18.7 Sets ... 597
A.18.8 The Generic Package Containers.Hashed_Sets ... 608
A.18.9 The Generic Package Containers.Ordered_Sets ... 618
A.18.10 The Generic Package Containers.Multiway_Trees ... 631
A.18.11 The Generic Package Containers.Indefinite_Vectors ... 667
A.18.12 The Generic Package Containers.Indefinite_Doubly_Linked_Lists ... 668
A.18.13 The Generic Package Containers.Indefinite_Hashed_Maps ... 669
A.18.14 The Generic Package Containers.Indefinite_Ordered_Maps ... 669
A.18.15 The Generic Package Containers.Indefinite_Hashed_Sets ... 670
A.18.16 The Generic Package Containers.Indefinite_Ordered_Sets ... 670
A.18.17 The Generic Package Containers.Indefinite_Multiway_Trees ... 670
A.18.18 The Generic Package Containers.Indefinite_Holders ... 671
A.18.19 The Generic Package Containers.Bounded_Vectors ... 676
A.18.20 The Generic Package Containers.Bounded_Doubly_Linked_Lists ... 677
A.18.21 The Generic Package Containers.Bounded_Hashed_Maps ... 679
A.18.22 The Generic Package Containers.Bounded_Ordered_Maps ... 680
A.18.23 The Generic Package Containers.Bounded_Hashed_Sets ... 682
A.18.24 The Generic Package Containers.Bounded_Ordered_Sets ... 684
A.18.25 The Generic Package Containers.Bounded_Multiway_Trees ... 685
A.18.26 Array Sorting ... 687
A.18.27 The Generic Package Containers.Synchronized_Queue_Interfaces ... 689
A.18.28 The Generic Package Containers.Unbounded_Synchronized_Queues ... 690
A.18.29 The Generic Package Containers.Bounded_Synchronized_Queues ... 690
A.18.30 The Generic Package Containers.Unbounded_Priority_Queues ... 691
A.18.31 The Generic Package Containers.Bounded_Priority_Queues ... 693
A.18.32 The Generic Package Containers.Bounded_Indefinite_Holders ... 694
A.18.33 Example of Container Use ... 694
A.19 The Package Locales ... 697

Annex B (normative) Interface to Other Languages ... 698
B.1 Interfacing Aspects ... 698
B.2 The Package Interfaces ... 701
B.3 Interfacing with C and C++ ... 702
B.3.1 The Package Interfaces.C.Strings ... 709
B.3.2 The Generic Package Interfaces.C.Pointers ... 712
B.3.3 Unchecked Union Types ... 714
B.4 Interfacing with COBOL ... 716
B.5 Interfacing with Fortran ... 722

Annex C (informative) Systems Programming ... 725
C.1 Access to Machine Operations ... 725
C.2 Required Representation Support ... 726
C.3 Interrupt Support ... 726
C.3.1 Protected Procedure Handlers ... 728
C.3.2 The Package Interrupts ... 730
C.4 Preelaboration Requirements ... 732
C.5 Aspect Discard_Names ... 732
C.6 Shared Variable Control ... 733
C.6.1 The Package System.Atomic_Operations ... 736
C.6.2 The Package System.Atomic_Operations.Exchange ... 736
C.6.3 The Package System.Atomic_Operations.Test_and_Set ... 737
C.6.4 The Package System.Atomic_Operations.Integer_Arithmetic ... 738

© ISO/IEC 2023 – All rights reserved

C.6.5 The Package System.Atomic_Operations.Modular_Arithmetic 739
C.7 Task Information 740
C.7.1 The Package Task_Identification 740
C.7.2 The Package Task_Attributes 742
C.7.3 The Package Task_Termination 744
Annex D (informative) Real-Time Systems 746
D.1 Task Priorities 746
D.2 Priority Scheduling 748
D.2.1 The Task Dispatching Model 748
D.2.2 Task Dispatching Pragmas 750
D.2.3 Preemptive Dispatching 751
D.2.4 Non-Preemptive Dispatching 752
D.2.5 Round Robin Dispatching 753
D.2.6 Earliest Deadline First Dispatching 754
D.3 Priority Ceiling Locking 757
D.4 Entry Queuing Policies 759
D.4.1 Admission Policies 761
D.5 Dynamic Priorities 761
D.5.1 Dynamic Priorities for Tasks 761
D.5.2 Dynamic Priorities for Protected Objects 762
D.6 Preemptive Abort 763
D.7 Tasking Restrictions 764
D.8 Monotonic Time 767
D.9 Delay Accuracy 770
D.10 Synchronous Task Control 771
D.10.1 Synchronous Barriers 772
D.11 Asynchronous Task Control 773
D.12 Other Optimizations and Determinism Rules 774
D.13 The Ravenscar and Jorvik Profiles 775
D.14 Execution Time 777
D.14.1 Execution Time Timers 779
D.14.2 Group Execution Time Budgets 780
D.14.3 Execution Time of Interrupt Handlers 783
D.15 Timing Events 783
D.16 Multiprocessor Implementation 785
D.16.1 Multiprocessor Dispatching Domains 786
Annex E (informative) Distributed Systems 789
E.1 Partitions 789
E.2 Categorization of Library Units 790
E.2.1 Shared Passive Library Units 791
E.2.2 Remote Types Library Units 792
E.2.3 Remote Call Interface Library Units 793
E.3 Consistency of a Distributed System 794
E.4 Remote Subprogram Calls 795
E.4.1 Asynchronous Remote Calls 796
E.4.2 Example of Use of a Remote Access-to-Class-Wide Type 797
E.5 Partition Communication Subsystem 798
Annex F (informative) Information Systems 801
F.1 Machine_Radix Attribute Definition Clause 801
F.2 The Package Decimal 801
F.3 Edited Output for Decimal Types 802
F.3.1 Picture String Formation 804
F.3.2 Edited Output Generation 807
F.3.3 The Package Text_IO.Editing 811

 © ISO/IEC 2023 – All rights reserved

F.3.4 The Package Wide_Text_IO.Editing ... 814
F.3.5 The Package Wide_Wide_Text_IO.Editing ... 814
Annex G (informative) Numerics ... 815
G.1 Complex Arithmetic ... 815
G.1.1 Complex Types ... 815
G.1.2 Complex Elementary Functions ... 819
G.1.3 Complex Input-Output ... 823
G.1.4 The Package Wide_Text_IO.Complex_IO ... 825
G.1.5 The Package Wide_Wide_Text_IO.Complex_IO ... 825
G.2 Numeric Performance Requirements ... 825
G.2.1 Model of Floating Point Arithmetic ... 826
G.2.2 Model-Oriented Attributes of Floating Point Types ... 827
G.2.3 Model of Fixed Point Arithmetic ... 828
G.2.4 Accuracy Requirements for the Elementary Functions ... 829
G.2.5 Performance Requirements for Random Number Generation ... 831
G.2.6 Accuracy Requirements for Complex Arithmetic ... 831
G.3 Vector and Matrix Manipulation ... 833
G.3.1 Real Vectors and Matrices ... 834
G.3.2 Complex Vectors and Matrices ... 838
Annex H (informative) High Integrity Systems ... 849
H.1 Pragma Normalize_Scalars ... 849
H.2 Documentation of Implementation Decisions ... 850
H.3 Reviewable Object Code ... 850
H.3.1 Pragma Reviewable ... 850
H.3.2 Pragma Inspection_Point ... 851
H.4 High Integrity Restrictions ... 852
H.4.1 Aspect No_Controlled_Parts ... 855
H.5 Pragma Detect_Blocking ... 855
H.6 Pragma Partition_Elaboration_Policy ... 855
H.7 Extensions to Global and Global'Class Aspects ... 856
H.7.1 The Use_Formal and Dispatching Aspects ... 857
Annex I (normative) Obsolescent Features ... 860
I.1 Renamings of Library Units ... 860
I.2 Allowed Replacements of Characters ... 860
I.3 Reduced Accuracy Subtypes ... 861
I.4 The Constrained Attribute ... 861
I.5 ASCII ... 862
I.6 Numeric_Error ... 862
I.7 At Clauses ... 862
I.7.1 Interrupt Entries ... 863
I.8 Mod Clauses ... 864
I.9 The Storage_Size Attribute ... 864
I.10 Specific Suppression of Checks ... 865
I.11 The Class Attribute of Untagged Incomplete Types ... 865
I.12 Pragma Interface ... 865
I.13 Dependence Restriction Identifiers ... 865
I.14 Character and Wide_Character Conversion Functions ... 866
I.15 Aspect-related Pragmas ... 866
I.15.1 Pragma Inline ... 867
I.15.2 Pragma No_Return ... 867
I.15.3 Pragma Pack ... 867
I.15.4 Pragma Storage_Size ... 868
I.15.5 Interfacing Pragmas ... 868
I.15.6 Pragma Unchecked_Union ... 869

© ISO/IEC 2023 – All rights reserved

I.15.7 Pragmas Interrupt_Handler and Attach_Handler 869
I.15.8 Shared Variable Pragmas 870
I.15.9 Pragma CPU 870
I.15.10 Pragma Dispatching_Domain 871
I.15.11 Pragmas Priority and Interrupt_Priority 871
I.15.12 Pragma Relative_Deadline 872
I.15.13 Pragma Asynchronous 872
I.15.14 Elaboration Control Pragmas 872
I.15.15 Distribution Pragmas 873
Annex J (informative) Language-Defined Aspects and Attributes 875
J.1 Language-Defined Aspects 875
J.2 Language-Defined Attributes 879
Annex K (informative) Language-Defined Pragmas 897
Annex L (informative) Summary of Documentation Requirements 899
L.1 Specific Documentation Requirements 899
L.2 Implementation-Defined Characteristics 901
L.3 Implementation Advice 907
Annex M (informative) Syntax Summary 916
M.1 Syntax Rules 916
M.2 Syntax Cross Reference 941
Annex N (informative) Language-Defined Entities 967
N.1 Language-Defined Packages 967
N.2 Language-Defined Types and Subtypes 970
N.3 Language-Defined Subprograms 974
N.4 Language-Defined Exceptions 987
N.5 Language-Defined Objects 988
Bibliography 993
Index 994

 © ISO/IEC 2023 – All rights reserved

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see https://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This fourth edition cancels and replaces the third edition (ISO/IEC 8652:2012), which has been technically revised. It also incorporates the Technical Corrigendum ISO/IEC 8652:2012/Cor.1:2016.

The main changes are as follows:

— improved support for parallel execution is provided via the introduction of parallel loops, parallel blocks, parallel container iteration, and parallel reduction;

— more precise specification of subprogram interfaces is supported via the new aspects Global, Global'Class, and Nonblocking. The Global aspects, in particular, help to determine whether two constructs can safely execute in parallel;

— Pre and Post aspects can now be specified for access-to-subprogram types and for generic formal subprograms; a postcondition for the default initialization of a type can be specified using the new Default_Initial_Condition aspect;

— the behavior of many predefined container operations is now more precisely specified by using pre- and postcondition specifications instead of English descriptions; a restricted (“stable”) view for most containers is introduced to support more efficient iteration;

— more flexible uses of static expressions are supported via the introduction of static expression functions along with fewer restrictions on static strings;

— the Image attribute is supported for nonscalar types, and a user-specifiable attribute Put_Image is provided, which determines the value of the Image attribute for a user-defined type;

© ISO/IEC 2023 – All rights reserved

— the use of numeric and string literals is generalized to allow their use with other categories of types, via the new aspects Integer_Literal, Real_Literal, and String_Literal;

— array and record aggregates are made more flexible: index parameters are allowed in an array aggregate to define the components as a function of their array index; discriminants can be defined more flexibly within an aggregate for a variant record type;

— new types of aggregates are provided: delta aggregates to allow the construction of a new object by incremental updates to an existing object; container aggregates to allow construction of an object of a container type by directly specifying its elements;

— a shorthand is provided, using the token '@', to refer to the target of an assignment statement in the expression defining its new value;

— declare expressions are provided that permit the definition and use of local constants or renamings, to allow a large expression to be simplified by defining common parts as named entities;

— support for lightweight iteration is added via the introduction of procedural iterators;

— support for the map-reduce programming strategy is added via the introduction of reduction expressions;

— for constructs that use iterators of any sort, a filter can be specified that restricts the elements produced by the iteration to those that satisfy the condition of the filter;

— predefined packages supporting arbitrary-precision integer and real arithmetic are provided;

— the Jorvik profile is introduced to support hard real-time applications that want to go beyond the restrictions of the Ravenscar profile.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

© ISO/IEC 2023 – All rights reserved

Introduction

Design Goals

Ada was originally designed with three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency. The 1995 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. Subsequent editions, including this fourth edition, have provided further flexibility and added more standardized packages within the framework provided by the 1995 revision.

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.

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.

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.

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. Parallel constructs were introduced to simplify making safe and efficient use of modern multicore architectures.

Language Summary

An Ada program is composed of one or more program units. Program units can 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 is visible to other units, and a body, containing the implementation details, which are not visible to other units. Most program units can be compiled separately.

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.

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

© ISO/IEC 2023 – All rights reserved

individual components. The text of a separately compiled program unit names the library units it requires.

Program Units

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 can read data, update variables, or produce some output. It can 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.

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.

Subprogram and package units can 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.

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

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 blocks the execution of the body until it evaluates to True. A protected unit can define a single protected object or a protected type permitting the creation of several similar objects.

Declarations and Statements

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.

The declarative part associates names with declared entities. For example, a name can 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.

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

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.

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.

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

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

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 can accept an entry call by executing a corresponding accept

 © ISO/IEC 2023 – All rights reserved

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. Various parallel constructs, including parallel loops and parallel blocks, support the initiation of multiple logical threads of control designed to execute in parallel when multiple processors are available.

Execution of a program unit can encounter error situations in which normal program execution cannot continue. For example, an arithmetic computation can exceed the maximum allowed value of a number, or an attempt can 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.

Data Types

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

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, Wide_Character, and Wide_Wide_Character are predefined.

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, Wide_String, and Wide_Wide_String are predefined.

Record, task, and protected types can 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.

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 can designate the same object, and components of one object can 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.

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.

From any type a new type can be defined by derivation. A type, together with its derivatives (both direct and indirect) form a derivation class. Class-wide operations can be defined that accept as a parameter an operand of any type in a derivation class. For record and private types, the derivatives can be extensions of the parent type. Types that support these object-oriented capabilities of class-wide operations and type extension are 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.

© ISO/IEC 2023 – All rights reserved

Interface types provide abstract models from which other interfaces and types can be composed and derived. This provides a reliable form of multiple inheritance. Interface types can also be implemented by task types and protected types thereby enabling concurrent programming and inheritance to be merged.

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.

Other Facilities

Aspect 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 must 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.

Aspect clauses can also be used to specify more abstract properties of program entities, such as the pre- and postconditions of a subprogram, or the invariant for a private type. Additional aspects are specifiable to allow user-defined types to use constructs of the language, such as literals, aggregates, or indexing, normally reserved for particular language-defined categories of types, such as numeric types, record types, or array types.

The predefined environment of the language provides for input-output and other capabilities 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.

The predefined standard library packages provide facilities such as string manipulation, containers of various kinds (vectors, lists, maps, etc.), mathematical functions, random number generation, and access to the execution environment.

The specialized annexes define further predefined library packages and facilities with emphasis on areas such as real-time scheduling, interrupt handling, distributed systems, numerical computation, and high-integrity systems.

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.

 © ISO/IEC 2023 – All rights reserved

Information technology — Programming Languages — Ada

1 Scope

This document specifies the form and meaning of programs written in Ada. Its purpose is to promote the portability of Ada programs to a variety of computing systems.

This document specifies:

— The form of a program written in Ada;

— The effect of translating and executing such a program;

— The manner in which program units can be combined to form Ada programs;

— The language-defined library units that a conforming implementation is required to supply;

— The permissible variations in conformance to the rules of this document, and the manner in which they are to be documented;

— Those violations of the requirements of this document that a conforming implementation is required to detect, and the effect of attempting to translate or execute a program containing such violations;

— Those violations of the requirements of this document that a conforming implementation is not required to detect.

This document does not specify:

— The means whereby a program written in Ada is transformed into object code executable by a processor;

— The means whereby translation or execution of programs is invoked and the executing units are controlled;

— The size or speed of the object code, or the relative execution speed of different language constructs;

— The form or contents of any listings produced by implementations; in particular, the form or contents of error or warning messages;

— The effect of unspecified execution;

— The size of a program or program unit that will exceed the capacity of a particular conforming implementation.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 639-3:2007, *Codes for the representation of names of languages — Part 3: Alpha-3 code for comprehensive coverage of languages*

ISO 3166-1:2020, *Codes for the representation of names of countries and their subdivisions — Part 1: Country codes*

ISO/IEC 10646:2020, *Information technology — Universal coded character set (UCS)*

© ISO/IEC 2023 – All rights reserved

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1 Types, objects, and their properties

3.1.1
abstract type
tagged type intended for use as an ancestor of other types, but which is not allowed to have objects of its own

3.1.2
access type
type that has values that designate aliased objects

Note 1 to entry: Access types correspond to “pointer types” or “reference types” in some other languages.

3.1.3
accessibility level
representation of the lifetime of an entity in terms of the level of dynamic nesting within which the entity is known to exist

3.1.4
aliased view
view of an object that can be designated by an access value

Note 1 to entry: Objects allocated by allocators are aliased. Objects can also be explicitly declared as aliased with the reserved word aliased. The Access attribute can be used to create an access value designating an aliased object.

3.1.5
ancestor of a type
type itself or, in the case of a type derived from other types, its parent type or one of its progenitor types or one of their ancestors

Note 1 to entry: Ancestor and descendant are inverse relationships.

3.1.6
array type
composite type whose components are all of the same type

3.1.7
aspect
specifiable property of an entity

Note 1 to entry: An aspect can be specified by an `aspect_specification` on the declaration of the entity. Some aspects can be queried via attributes.

3.1.8
attribute
characteristic or property of an entity that can be queried, and in some cases specified

3.1.9
category of types
set of types with one or more common properties, such as primitive operations

Note 1 to entry: A category of types that is closed under derivation is also known as a class.

© ISO/IEC 2023 – All rights reserved