INTERNATIONAL STANDARD

ISO/IEC 1539-1

Third edition
2010-10-15

Information technology — Programming languages — Fortran —

Part 1: Base language

Technologies de l'information — Langages de programmation — Fortran —

Partie 1: Langage de base

Reference number
ISO/IEC 1539-1:2010(E)

Documents
Document Sharing Hub
© ISO/IEC 2010

Full document details: ISO_IEC 1539-1_2010

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2010

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

Contents

Foreword . . . xiv

Introduction . . . xv

1 Overview . . . 1
1.1 Scope . . . 1
1.2 Normative references . . . 1
1.3 Terms and definitions . . . 2
1.4 Notation, symbols and abbreviated terms . . . 21
1.4.1 Syntax rules . . . 21
1.4.2 Constraints . . . 22
1.4.3 Assumed syntax rules . . . 22
1.4.4 Syntax conventions and characteristics . . . 22
1.4.5 Text conventions . . . 23
1.5 Conformance . . . 23
1.6 Compatibility . . . 24
1.6.1 New intrinsic procedures . . . 24
1.6.2 Fortran 2003 compatibility . . . 24
1.6.3 Fortran 95 compatibility . . . 24
1.6.4 Fortran 90 compatibility . . . 24
1.6.5 FORTRAN 77 compatibility . . . 25
1.7 Deleted and obsolescent features . . . 25
1.7.1 General . . . 25
1.7.2 Nature of deleted features . . . 25
1.7.3 Nature of obsolescent features . . . 26

2 Fortran concepts . . . 27
2.1 High level syntax . . . 27
2.2 Program unit concepts . . . 30
2.2.1 Program units and scoping units . . . 30
2.2.2 Program . . . 30
2.2.3 Procedure . . . 30
2.2.4 Module . . . 31
2.2.5 Submodule . . . 31
2.3 Execution concepts . . . 31
2.3.1 Statement classification . . . 31
2.3.2 Statement order . . . 31
2.3.3 The END statement . . . 32
2.3.4 Program execution . . . 32
2.3.5 Execution sequence . . . 33
2.4 Data concepts . . . 34
2.4.1 Type . . . 34
2.4.2 Data value . . . 34
2.4.3 Data entity . . . 34
2.4.4 Definition of objects and pointers . . . 36
2.4.5 Reference . . . 36
2.4.6 Array . . . 36
2.4.7 Coarray . . . 37

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

2.4.8 Pointer . . . 37
2.4.9 Allocatable variables . . . 37
2.4.10 Storage . . . 38
2.5 Fundamental concepts . . . 38
2.5.1 Names and designators . . . 38
2.5.2 Statement keyword . . . 38
2.5.3 Other keywords . . . 38
2.5.4 Association . . . 38
2.5.5 Intrinsic . . . 38
2.5.6 Operator . . . 39
2.5.7 Companion processors . . . 39

3 Lexical tokens and source form . . . 41
3.1 Processor character set . . . 41
3.1.1 Characters . . . 41
3.1.2 Letters . . . 41
3.1.3 Digits . . . 41
3.1.4 Underscore . . . 41
3.1.5 Special characters . . . 41
3.1.6 Other characters . . . 42
3.2 Low-level syntax . . . 42
3.2.1 Tokens . . . 42
3.2.2 Names . . . 42
3.2.3 Constants . . . 43
3.2.4 Operators . . . 43
3.2.5 Statement labels . . . 44
3.2.6 Delimiters . . . 45
3.3 Source form . . . 45
3.3.1 Program units, statements, and lines . . . 45
3.3.2 Free source form . . . 45
3.3.3 Fixed source form . . . 47
3.4 Including source text . . . 47

4 Types . . . 49
4.1 The concept of type . . . 49
4.1.1 General . . . 49
4.1.2 Set of values . . . 49
4.1.3 Constants . . . 49
4.1.4 Operations . . . 49
4.2 Type parameters . . . 49
4.3 Relationship of types and values to objects . . . 51
4.3.1 Type specifiers and type compatibility . . . 51
4.4 Intrinsic types . . . 52
4.4.1 Classification and specification . . . 52
4.4.2 Numeric intrinsic types . . . 53
4.4.3 Character type . . . 56
4.4.4 Logical type . . . 60
4.5 Derived types . . . 60
4.5.1 Derived type concepts . . . 60
4.5.2 Derived-type definition . . . 61
4.5.3 Derived-type parameters . . . 64
4.5.4 Components . . . 66
4.5.5 Type-bound procedures . . . 73
4.5.6 Final subroutines . . . 75
4.5.7 Type extension . . . 77
4.5.8 Derived-type values . . . 79

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

4.5.9 Derived-type specifier . . . 79
4.5.10 Construction of derived-type values . . . 80
4.5.11 Derived-type operations and assignment . . . 82
4.6 Enumerations and enumerators . . . 82
4.7 Binary, octal, and hexadecimal literal constants . . . 83
4.8 Construction of array values . . . 84

5 Attribute declarations and specifications . . . 87
5.1 General . . . 87
5.2 Type declaration statements . . . 87
5.2.1 Syntax . . . 87
5.2.2 Automatic data objects . . . 88
5.2.3 Initialization . . . 89
5.2.4 Examples of type declaration statements . . . 89
5.3 Attributes . . . 89
5.3.1 Constraints . . . 89
5.3.2 Accessibility attribute . . . 89
5.3.3 ALLOCATABLE attribute . . . 90
5.3.4 ASYNCHRONOUS attribute . . . 90
5.3.5 BIND attribute for data entities . . . 90
5.3.6 CODIMENSION attribute . . . 91
5.3.7 CONTIGUOUS attribute . . . 93
5.3.8 DIMENSION attribute . . . 94
5.3.9 EXTERNAL attribute . . . 96
5.3.10 INTENT attribute . . . 97
5.3.11 INTRINSIC attribute . . . 98
5.3.12 OPTIONAL attribute . . . 99
5.3.13 PARAMETER attribute . . . 99
5.3.14 POINTER attribute . . . 99
5.3.15 PROTECTED attribute . . . 100
5.3.16 SAVE attribute . . . 100
5.3.17 TARGET attribute . . . 101
5.3.18 VALUE attribute . . . 101
5.3.19 VOLATILE attribute . . . 102
5.4 Attribute specification statements . . . 102
5.4.1 Accessibility statement . . . 102
5.4.2 ALLOCATABLE statement . . . 103
5.4.3 ASYNCHRONOUS statement . . . 103
5.4.4 BIND statement . . . 103
5.4.5 CODIMENSION statement . . . 103
5.4.6 CONTIGUOUS statement . . . 104
5.4.7 DATA statement . . . 104
5.4.8 DIMENSION statement . . . 106
5.4.9 INTENT statement . . . 106
5.4.10 OPTIONAL statement . . . 107
5.4.11 PARAMETER statement . . . 107
5.4.12 POINTER statement . . . 107
5.4.13 PROTECTED statement . . . 107
5.4.14 SAVE statement . . . 108
5.4.15 TARGET statement . . . 108
5.4.16 VALUE statement . . . 108
5.4.17 VOLATILE statement . . . 108
5.5 IMPLICIT statement . . . 108
5.6 NAMELIST statement . . . 111
5.7 Storage association of data objects . . . 112
5.7.1 EQUIVALENCE statement . . . 112

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

5.7.2 COMMON statement . . . 114
5.7.3 Restrictions on common and equivalence . . . 116

6 Use of data objects . . . 117
6.1 Designator . . . 117
6.2 Variable . . . 117
6.3 Constants . . . 118
6.4 Scalars . . . 118
6.4.1 Substrings . . . 118
6.4.2 Structure components . . . 118
6.4.3 Coindexed named objects . . . 120
6.4.4 Complex parts . . . 120
6.4.5 Type parameter inquiry . . . 120
6.5 Arrays . . . 121
6.5.1 Order of reference . . . 121
6.5.2 Whole arrays . . . 121
6.5.3 Array elements and array sections . . . 121
6.5.4 Simply contiguous array designators . . . 124
6.6 Image selectors . . . 125
6.7 Dynamic association . . . 126
6.7.1 ALLOCATE statement . . . 126
6.7.2 NULLIFY statement . . . 129
6.7.3 DEALLOCATE statement . . . 130
6.7.4 STAT= specifier . . . 132
6.7.5 ERRMSG= specifier . . . 132

7 Expressions and assignment . . . 133
7.1 Expressions . . . 133
7.1.1 General . . . 133
7.1.2 Form of an expression . . . 133
7.1.3 Precedence of operators . . . 137
7.1.4 Evaluation of operations . . . 139
7.1.5 Intrinsic operations . . . 139
7.1.6 Defined operations . . . 146
7.1.7 Evaluation of operands . . . 147
7.1.8 Integrity of parentheses . . . 148
7.1.9 Type, type parameters, and shape of an expression . . . 148
7.1.10 Conformability rules for elemental operations . . . 149
7.1.11 Specification expression . . . 150
7.1.12 Constant expression . . . 151
7.2 Assignment . . . 153
7.2.1 Assignment statement . . . 153
7.2.2 Pointer assignment . . . 157
7.2.3 Masked array assignment – WHERE . . . 161
7.2.4 FORALL . . . 163

8 Execution control . . . 169
8.1 Executable constructs containing blocks . . . 169
8.1.1 General . . . 169
8.1.2 Rules governing blocks . . . 169
8.1.3 ASSOCIATE construct . . . 170
8.1.4 BLOCK construct . . . 171
8.1.5 CRITICAL construct . . . 173
8.1.6 DO construct . . . 174
8.1.7 IF construct and statement . . . 179
8.1.8 SELECT CASE construct . . . 181

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

8.1.9 SELECT TYPE construct . . . 184
8.1.10 EXIT statement . . . 186
8.2 Branching . . . 186
8.2.1 Branch concepts . . . 186
8.2.2 GO TO statement . . . 187
8.2.3 Computed GO TO statement . . . 187
8.2.4 Arithmetic IF statement . . . 187
8.3 CONTINUE statement . . . 187
8.4 STOP and ERROR STOP statements . . . 187
8.5 Image execution control . . . 188
8.5.1 Image control statements . . . 188
8.5.2 Segments . . . 189
8.5.3 SYNC ALL statement . . . 190
8.5.4 SYNC IMAGES statement . . . 190
8.5.5 SYNC MEMORY statement . . . 192
8.5.6 LOCK and UNLOCK statements . . . 193
8.5.7 STAT= and ERRMSG= specifiers in image control statements . . . 195

9 Input/output statements . . . 197
9.1 Input/output concepts . . . 197
9.2 Records . . . 197
9.2.1 General . . . 197
9.2.2 Formatted record . . . 197
9.2.3 Unformatted record . . . 197
9.2.4 Endfile record . . . 198
9.3 External files . . . 198
9.3.1 Basic concepts . . . 198
9.3.2 File existence . . . 198
9.3.3 File access . . . 199
9.3.4 File position . . . 201
9.3.5 File storage units . . . 202
9.4 Internal files . . . 203
9.5 File connection . . . 203
9.5.1 Referring to a file . . . 203
9.5.2 Connection modes . . . 204
9.5.3 Unit existence . . . 205
9.5.4 Connection of a file to a unit . . . 205
9.5.5 Preconnection . . . 206
9.5.6 OPEN statement . . . 206
9.5.7 CLOSE statement . . . 210
9.6 Data transfer statements . . . 211
9.6.1 General . . . 211
9.6.2 Control information list . . . 212
9.6.3 Data transfer input/output list . . . 217
9.6.4 Execution of a data transfer input/output statement . . . 219
9.6.5 Termination of data transfer statements . . . 229
9.7 Waiting on pending data transfer . . . 230
9.7.1 Wait operation . . . 230
9.7.2 WAIT statement . . . 230
9.8 File positioning statements . . . 231
9.8.1 Syntax . . . 231
9.8.2 BACKSPACE statement . . . 232
9.8.3 ENDFILE statement . . . 232
9.8.4 REWIND statement . . . 232
9.9 FLUSH statement . . . 233
9.10 File inquiry statement . . . 233

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

9.10.1 Forms of the INQUIRE statement . . . 233
9.10.2 Inquiry specifiers . . . 234
9.10.3 Inquire by output list . . . 240
9.11 Error, end-of-record, and end-of-file conditions . . . 240
9.11.1 General . . . 240
9.11.2 Error conditions and the ERR= specifier . . . 240
9.11.3 End-of-file condition and the END= specifier . . . 241
9.11.4 End-of-record condition and the EOR= specifier . . . 241
9.11.5 IOSTAT= specifier . . . 242
9.11.6 IOMSG= specifier . . . 242
9.12 Restrictions on input/output statements . . . 242

10 Input/output editing . . . 245
10.1 Format specifications . . . 245
10.2 Explicit format specification methods . . . 245
10.2.1 FORMAT statement . . . 245
10.2.2 Character format specification . . . 245
10.3 Form of a format item list . . . 246
10.3.1 Syntax . . . 246
10.3.2 Edit descriptors . . . 246
10.3.3 Fields . . . 248
10.4 Interaction between input/output list and format . . . 248
10.5 Positioning by format control . . . 250
10.6 Decimal symbol . . . 250
10.7 Data edit descriptors . . . 250
10.7.1 General . . . 250
10.7.2 Numeric editing . . . 251
10.7.3 Logical editing . . . 256
10.7.4 Character editing . . . 257
10.7.5 Generalized editing . . . 257
10.7.6 User-defined derived-type editing . . . 259
10.8 Control edit descriptors . . . 259
10.8.1 Position editing . . . 259
10.8.2 Slash editing . . . 260
10.8.3 Colon editing . . . 260
10.8.4 SS, SP, and S editing . . . 261
10.8.5 P editing . . . 261
10.8.6 BN and BZ editing . . . 261
10.8.7 RU, RD, RZ, RN, RC, and RP editing . . . 261
10.8.8 DC and DP editing . . . 262
10.9 Character string edit descriptors . . . 262
10.10 List-directed formatting . . . 262
10.10.1 General . . . 262
10.10.2 Values and value separators . . . 262
10.10.3 List-directed input . . . 263
10.10.4 List-directed output . . . 265
10.11 Namelist formatting . . . 266
10.11.1 General . . . 266
10.11.2 Name-value subsequences . . . 266
10.11.3 Namelist input . . . 267
10.11.4 Namelist output . . . 270

11 Program units . . . 271
11.1 Main program . . . 271
11.2 Modules . . . 271
11.2.1 General . . . 271

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

11.2.2 The USE statement and use association . . . 272
11.2.3 Submodules . . . 275
11.3 Block data program units . . . 275

12 Procedures . . . 277
12.1 Concepts . . . 277
12.2 Procedure classifications . . . 277
12.2.1 Procedure classification by reference . . . 277
12.2.2 Procedure classification by means of definition . . . 277
12.3 Characteristics . . . 278
12.3.1 Characteristics of procedures . . . 278
12.3.2 Characteristics of dummy arguments . . . 278
12.3.3 Characteristics of function results . . . 278
12.4 Procedure interface . . . 279
12.4.1 Interface and abstract interface . . . 279
12.4.2 Implicit and explicit interfaces . . . 279
12.4.3 Specification of the procedure interface . . . 280
12.5 Procedure reference . . . 289
12.5.1 Syntax of a procedure reference . . . 289
12.5.2 Actual arguments, dummy arguments, and argument association . . . 291
12.5.3 Function reference . . . 302
12.5.4 Subroutine reference . . . 302
12.5.5 Resolving named procedure references . . . 302
12.5.6 Resolving type-bound procedure references . . . 305
12.6 Procedure definition . . . 305
12.6.1 Intrinsic procedure definition . . . 305
12.6.2 Procedures defined by subprograms . . . 305
12.6.3 Definition and invocation of procedures by means other than Fortran . . . 311
12.6.4 Statement function . . . 311
12.7 Pure procedures . . . 312
12.8 Elemental procedures . . . 313
12.8.1 Elemental procedure declaration and interface . . . 313
12.8.2 Elemental function actual arguments and results . . . 314
12.8.3 Elemental subroutine actual arguments . . . 314

13 Intrinsic procedures and modules . . . 315
13.1 Classes of intrinsic procedures . . . 315
13.2 Arguments to intrinsic procedures . . . 315
13.2.1 General rules . . . 315
13.2.2 The shape of array arguments . . . 316
13.2.3 Mask arguments . . . 316
13.2.4 DIM arguments and reduction functions . . . 316
13.3 Bit model . . . 317
13.3.1 General . . . 317
13.3.2 Bit sequence comparisons . . . 317
13.3.3 Bit sequences as arguments to INT and REAL . . . 317
13.4 Numeric models . . . 318
13.5 Standard generic intrinsic procedures . . . 318
13.6 Specific names for standard intrinsic functions . . . 323
13.7 Specifications of the standard intrinsic procedures . . . 325
13.7.1 General . . . 325
13.8 Standard modules . . . 396
13.8.1 General . . . 396
13.8.2 The ISO_FORTRAN_ENV intrinsic module . . . 397

14 Exceptions and IEEE arithmetic . . . 401

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

14.1 General . . . 401
14.2 Derived types and constants defined in the modules . . . 402
14.3 The exceptions . . . 403
14.4 The rounding modes . . . 404
14.5 Underflow mode . . . 405
14.6 Halting . . . 405
14.7 The floating-point status . . . 405
14.8 Exceptional values . . . 405
14.9 IEEE arithmetic . . . 406
14.10 Summary of the procedures . . . 407
14.11 Specifications of the procedures . . . 408
14.11.1 General . . . 408
14.12 Examples . . . 422

15 Interoperability with C . . . 425
15.1 General . . . 425
15.2 The ISO_C_BINDING intrinsic module . . . 425
15.2.1 Summary of contents . . . 425
15.2.2 Named constants and derived types in the module . . . 425
15.2.3 Procedures in the module . . . 426
15.3 Interoperability between Fortran and C entities . . . 429
15.3.1 General . . . 429
15.3.2 Interoperability of intrinsic types . . . 429
15.3.3 Interoperability with C pointer types . . . 431
15.3.4 Interoperability of derived types and C struct types . . . 431
15.3.5 Interoperability of scalar variables . . . 432
15.3.6 Interoperability of array variables . . . 432
15.3.7 Interoperability of procedures and procedure interfaces . . . 433
15.4 Interoperation with C global variables . . . 435
15.4.1 General . . . 435
15.4.2 Binding labels for common blocks and variables . . . 436
15.5 Interoperation with C functions . . . 436
15.5.1 Definition and reference of interoperable procedures . . . 436
15.5.2 Binding labels for procedures . . . 437
15.5.3 Exceptions and IEEE arithmetic procedures . . . 437

16 Scope, association, and definition . . . 439
16.1 Scopes, identifiers, and entities . . . 439
16.2 Global identifiers . . . 439
16.3 Local identifiers . . . 440
16.3.1 Classes of local identifiers . . . 440
16.3.2 Local identifiers that are the same as common block names . . . 441
16.3.3 Function results . . . 441
16.3.4 Components, type parameters, and bindings . . . 441
16.3.5 Argument keywords . . . 442
16.4 Statement and construct entities . . . 442
16.5 Association . . . 443
16.5.1 Name association . . . 443
16.5.2 Pointer association . . . 446
16.5.3 Storage association . . . 449
16.5.4 Inheritance association . . . 451
16.5.5 Establishing associations . . . 451
16.6 Definition and undefinition of variables . . . 452
16.6.1 Definition of objects and subobjects . . . 452
16.6.2 Variables that are always defined . . . 452
16.6.3 Variables that are initially defined . . . 453

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

16.6.4 Variables that are initially undefined . . . 453
16.6.5 Events that cause variables to become defined . . . 453
16.6.6 Events that cause variables to become undefined . . . 455
16.6.7 Variable definition context . . . 456
16.6.8 Pointer association context . . . 457

Annex A (informative) Processor Dependencies . . . 459
A.1 Unspecified Items . . . 459
A.2 Processor Dependencies . . . 459

Annex B (informative) Deleted and obsolescent features . . . 463
B.1 Deleted features . . . 463
B.2 Obsolescent features . . . 464
B.2.1 General . . . 464
B.2.2 Alternate return . . . 464
B.2.3 Computed GO TO statement . . . 464
B.2.4 Statement functions . . . 464
B.2.5 DATA statements among executables . . . 465
B.2.6 Assumed character length functions . . . 465
B.2.7 Fixed form source . . . 465
B.2.8 CHARACTER* form of CHARACTER declaration . . . 465
B.2.9 ENTRY statements . . . 465

Annex C (informative) Extended notes . . . 467
C.1 Clause 4 notes . . . 467
C.1.1 Selection of the approximation methods (4.4.2.3) . . . 467
C.1.2 Type extension and component accessibility (4.5.2.2, 4.5.4) . . . 467
C.1.3 Generic type-bound procedures (4.5.5) . . . 468
C.1.4 Abstract types (4.5.7.1) . . . 469
C.1.5 Pointers (4.5.4.4, 5.3.14) . . . 470
C.1.6 Structure constructors and generic names (4.5.10) . . . 471
C.1.7 Final subroutines (4.5.6, 4.5.6.2, 4.5.6.3, 4.5.6.4) . . . 473
C.2 Clause 5 notes . . . 474
C.2.1 The POINTER attribute (5.3.14) . . . 474
C.2.2 The TARGET attribute (5.3.17) . . . 475
C.2.3 The VOLATILE attribute (5.3.19) . . . 476
C.3 Clause 6 notes . . . 476
C.3.1 Structure components (6.4.2) . . . 476
C.3.2 Allocation with dynamic type (6.7.1) . . . 478
C.3.3 Pointer allocation and association (6.7.1, 16.5.2) . . . 478
C.4 Clause 7 notes . . . 479
C.4.1 Character assignment (7.2.1.3) . . . 479
C.4.2 Evaluation of function references (7.1.7) . . . 479
C.4.3 Pointers in expressions (7.1.9.2) . . . 480
C.4.4 Pointers in variable-definition contexts (7.2.1.3, 16.6.7) . . . 480
C.4.5 Examples of FORALL constructs (7.2.4) . . . 480
C.4.6 Examples of FORALL statements (7.2.4.3) . . . 482
C.5 Clause 8 notes . . . 483
C.5.1 The SELECT CASE construct (8.1.8) . . . 483
C.5.2 Loop control (8.1.6) . . . 483
C.5.3 Examples of DO constructs (8.1.6) . . . 483
C.5.4 Examples of invalid DO constructs (8.1.6) . . . 485
C.6 Clause 9 notes . . . 486
C.6.1 External files (9.3) . . . 486
C.6.2 Nonadvancing input/output (9.3.4.2) . . . 487
C.6.3 OPEN statement (9.5.6) . . . 488

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

C.6.4 Connection properties (9.5.4) . . . 490
C.6.5 CLOSE statement (9.5.7) . . . 490
C.6.6 Asynchronous input/output (9.6.2.5) . . . 490
C.7 Clause 10 notes . . . 492
C.7.1 Number of records (10.4, 10.5, 10.8.2) . . . 492
C.7.2 List-directed input (10.10.3) . . . 492
C.8 Clause 11 notes . . . 493
C.8.1 Main program and block data program unit (11.1, 11.3) . . . 493
C.8.2 Dependent compilation (11.2) . . . 493
C.8.3 Examples of the use of modules (11.2.1) . . . 495
C.8.4 Modules with submodules (11.2.3) . . . 501
C.9 Clause 12 notes . . . 505
C.9.1 Portability problems with external procedures (12.4.3.5) . . . 505
C.9.2 Procedures defined by means other than Fortran (12.6.3) . . . 505
C.9.3 Abstract interfaces (12.4) and procedure pointer components (4.5) . . . 505
C.9.4 Pointers and targets as arguments (12.5.2.4, 12.5.2.6, 12.5.2.7) . . . 507
C.9.5 Polymorphic Argument Association (12.5.2.9) . . . 509
C.9.6 Rules ensuring unambiguous generics (12.4.3.4.5) . . . 510
C.10 Clause 13 notes . . . 514
C.10.1 Module for THIS_IMAGE and IMAGE_INDEX . . . 514
C.11 Clause 15 notes . . . 515
C.11.1 Runtime environments (15.1) . . . 515
C.11.2 Example of Fortran calling C (15.3) . . . 515
C.11.3 Example of C calling Fortran (15.3) . . . 516
C.11.4 Example of calling C functions with noninteroperable data (15.5) . . . 518
C.11.5 Example of opaque communication between C and Fortran (15.3) . . . 518
C.12 Clause 16 notes . . . 520
C.12.1 Examples of host association (16.5.1.4) . . . 520
C.13 Array feature notes . . . 521
C.13.1 Summary of features (2.4.6) . . . 521
C.13.2 Examples (6.5) . . . 522
C.13.3 FORmula TRANslation and array processing (6.5) . . . 526
C.13.4 Logical queries (13.7.10, 13.7.13, 13.7.41, 13.7.109, 13.7.115 13.7.161) . . . 528
C.13.5 Parallel computations (7.1.2) . . . 528
C.13.6 Example of element-by-element computation (6.5.3) . . . 528

Annex D (informative) Syntax rules and constraints . . . 531
D.1 Extract of all syntax rules and constraints . . . 531
D.2 Syntax rule cross-reference . . . 572

Index . . . 585

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

List of Tables

2.1 Requirements on statement ordering . . . 31
2.2 Statements allowed in scoping units . . . 32

3.1 Special characters . . . 42

6.1 Subscript order value . . . 122

7.1 Categories of operations and relative precedence . . . 137
7.2 Type of operands and results for intrinsic operators . . . 140
7.3 Interpretation of the numeric intrinsic operators . . . 141
7.4 Interpretation of the character intrinsic operator // . . . 143
7.5 Interpretation of the logical intrinsic operators . . . 144
7.6 The values of operations involving logical intrinsic operators . . . 144
7.7 Interpretation of the relational intrinsic operators . . . 145
7.8 Type conformance for the intrinsic assignment statement . . . 153
7.9 Numeric conversion and the assignment statement . . . 155

10.1 E and D exponent forms . . . 253
10.2 EN exponent forms . . . 254
10.3 ES exponent forms . . . 255

13.1 Standard generic intrinsic procedure summary . . . 319
13.2 Characteristics of the result of NULL ( ) . . . 375

14.1 IEEE_ARITHMETIC module procedure summary . . . 407
14.2 IEEE_EXCEPTIONS module procedure summary . . . 408

15.1 Names of C characters with special semantics . . . 426
15.2 Interoperability between Fortran and C types . . . 430

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

Foreword

1 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 nongovernmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

2 International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

3 The main task of the joint technical committee is to prepare International Standards. 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.

4 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.

5 ISO/IEC 1539-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

6 This third edition cancels and replaces the second edition (ISO/IEC 1539-1:2004), which has been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 1539-1:2004/Cor. 1:2006, ISO/IEC 1539-1:2004/Cor. 2:2007, ISO/IEC 1539-1:2004/Cor. 3:2008, and ISO/IEC 1539-1:2004/Cor. 4:2009, and the Technical Report ISO/IEC TR 19767:2005.

7 ISO/IEC 1539 consists of the following parts, under the general title *Information technology — Programming languages — Fortran*:

8 — *Part 1: Base language*

9 — *Part 2: Varying length character strings*

10 — *Part 3: Conditional compilation*

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

Introduction

1 This part of ISO/IEC 1539 comprises the specification of the base Fortran language, informally known as Fortran 2008. With the limitations noted in 1.6.2, the syntax and semantics of Fortran 2003 are contained entirely within Fortran 2008. Therefore, any standard-conforming Fortran 2003 program not affected by such limitations is a standard-conforming Fortran 2008 program. New features of Fortran 2008 can be compatibly incorporated into such Fortran 2003 programs, with any exceptions indicated in the text of this part of ISO/IEC 1539.

2 Fortran 2008 contains several extensions to Fortran 2003; some of these are listed below.

- Module enhancements:
  Submodules provide additional structuring facilities for modules. Data objects and procedure pointers declared in a module implicitly have the SAVE attribute.
- Parallel execution:
  Coarrays and synchronization constructs support parallel programming using a single program multiple data (SPMD) model.
- Performance enhancements:
  The DO CONCURRENT construct provides a means for the program to specify that individual loop iterations have no interdependencies. The CONTIGUOUS attribute provides a means for the program to specify restrictions on the storage layout of pointer targets and assumed-shape dummy arguments.
- Data declaration:
  The maximum rank has been increased to 15. A processor is required to support at least one kind of integer with a range of at least 18 decimal digits. An allocatable component can be of recursive type. A named constant array's shape can be implied by its value. A pointer can be initially associated with a target. Subscripts and nested implied-do limits inside a *data-implied-do* can be any constant expression instead of being limited to combinations of constants, implied-do variables, and intrinsic operations. A FORALL index variable can have its type and kind explicitly declared within the construct. The TYPE keyword can be used to declare entities of intrinsic type. Multiple type-bound procedures can be declared in a single type-bound procedure statement.
- Data usage and computation:
  A structure constructor can omit the value for an allocatable component. SOURCE= in an ALLOCATE statement can give an array variable the bounds as well as the value of an expression. MOLD= in an ALLOCATE statement can give a polymorphic variable the shape, type, and type parameters of an expression without copying the value. The real and imaginary parts of a complex entity can be accessed independently with a component-like syntax. Intrinsic assignment to an allocatable polymorphic variable is allowed. A pointer function reference can denote a variable in any variable definition context. Some restrictions on the use of dummy arguments in elemental subprograms have been removed.
- Input/output:
  NEWUNIT= in an OPEN statement automatically selects a unit number that does not interfere with other unit numbers selected by the program. The G0 edit descriptor and unlimited format control ease writing output in comma-separated-value (CSV) format. Recursive data transfers are allowed on distinct units.
- Execution control:
  The BLOCK construct can contain declarations of objects with construct scope. The EXIT statement can transfer control from within more named executable constructs. The STOP statement has been changed to accept a constant expression instead of merely a literal constant, and to encourage the processor to provide the integer stop code (if it appears) as a termination status (where that makes sense). The ERROR STOP statement initiates error termination.
- Intrinsic procedures:
  - The intrinsic functions ACOS, ASIN, ATAN, COSH, SINH, TAN, and TANH can have arguments of type complex.
  - The new intrinsic functions ACOSH, ASINH, and ATANH calculate the inverse hyperbolic cosine, sine, and tangent respectively.
  - The intrinsic function ATAN2 can be referenced by the name ATAN.
  - The new intrinsic subroutines ATOMIC_DEFINE and ATOMIC_REF define and reference a variable

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

atomically.
- The new intrinsic functions BESSEL_J0, BESSEL_J1, BESSEL_JN, BESSEL_Y0, BESSEL_Y1, and BESSEL_YN calculate Bessel functions.
- The new intrinsic functions BGE, BGT, BLE, and BLT perform bitwise comparisons.
- The new intrinsic functions DSHIFTL and DSHIFTR calculate combined left and right shifts.
- The new intrinsic functions ERF, ERFC, and ERFC_SCALED calculate the error function and its complement.
- The new intrinsic subroutine EXECUTE_COMMAND_LINE allows a program to start another program.
- The new intrinsic function FINDLOC searches an array for a value.
- The intrinsic functions LGE, LGT, LLE, and LLT can have arguments of ASCII kind.
- The new intrinsic functions GAMMA and LOG_GAMMA calculate the gamma function and its log.
- The new intrinsic function HYPOT calculates the Euclidean distance.
- The new intrinsic functions IALL, IANY, and IPARITY reduce an array with the bitwise AND, bitwise OR, and bitwise exclusive OR functions respectively.
- The new intrinsic function IMAGE_INDEX converts cosubscripts to an image index.
- The new intrinsic functions LCOBOUND and UCOBOUND return the cobounds of a coarray.
- The new intrinsic functions LEADZ and TRAILZ return the number of leading and trailing zero bits in an integer.
- The new intrinsic functions MASKL and MASKR return simple left and right justified masks.
- A BACK= argument has been added to the intrinsic functions MAXLOC and MINLOC.
- The new intrinsic function MERGE_BITS performs a bitwise merge using a mask.
- The new intrinsic function NORM2 calculates the L_2 norm of an array.
- The new intrinsic function NUM_IMAGES returns the number of images.
- The new intrinsic function PARITY reduces an array with the .NEQV. operation.
- The new intrinsic functions POPCNT and POPPAR return the number of 1 bits of an integer and its parity.
- A RADIX= argument has been added to the intrinsic function SELECTED_REAL_KIND.
- The new intrinsic functions SHIFTA, SHIFTL and SHIFTR perform shift operations.
- The new intrinsic function STORAGE_SIZE returns the size of an array element in bits.
- The new intrinsic function THIS_IMAGE returns the index of this image or cosubscripts for it.

• Intrinsic modules:
The functions COMPILER_VERSION and COMPILER_OPTIONS in the intrinsic module ISO_FORTRAN_ENV return information about the program translation phase. Named constants for selecting kind values have been added to the intrinsic module ISO_FORTRAN_ENV. The function C_SIZEOF in the intrinsic module ISO_C_BINDING returns the size of an array element in bytes. A RADIX= argument has been added to the function IEEE_SELECTED_REAL_KIND in the intrinsic module IEEE_ARITHMETIC.

• Programs and procedures:
An empty CONTAINS section is allowed. An internal procedure can be used as an actual argument or procedure pointer target. ALLOCATABLE and POINTER attributes are used in generic resolution. Procedureness of a dummy argument is used in generic resolution. An actual argument with the TARGET attribute can correspond to a dummy pointer. A null pointer or unallocated allocatable can be used to denote an absent nonallocatable nonpointer optional argument. An impure elemental procedure processes array arguments in array element order. The FUNCTION and SUBROUTINE keywords can be omitted from the END statement for a module or internal subprogram. A line in the program is permitted to begin with a semicolon.

3 Additionally, the ENTRY feature present in FORTRAN 77 onwards is now deemed to be obsolescent by this part of ISO/IEC 1539.

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

4 This part of ISO/IEC 1539 is organized in 16 clauses, dealing with 8 conceptual areas. These 8 areas, and the clauses in which they are treated, are:

High/low level concepts	Clauses 1, 2, 3
Data concepts	Clauses 4, 5, 6
Computations	Clauses 7, 13, 14
Execution control	Clause 8
Input/output	Clauses 9, 10
Program units	Clauses 11, 12
Interoperability with C	Clause 15
Scoping and association rules	Clause 16

5 It also contains the following nonnormative material:

Processor dependencies	Annex A
Deleted and obsolescent features	Annex B
Extended notes	Annex C
Syntax rules	Annex D
Index	Index

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

Information technology — Programming languages — Fortran —

Part 1:

Base language

1 Overview

1.1 Scope

1 This part of ISO/IEC 1539 specifies the form and establishes the interpretation of programs expressed in the base Fortran language. The purpose of this part of ISO/IEC 1539 is to promote portability, reliability, maintainability, and efficient execution of Fortran programs for use on a variety of computing systems.

2 This part of ISO/IEC 1539 specifies

- the forms that a program written in the Fortran language may take,
- the rules for interpreting the meaning of a program and its data,
- the form of the input data to be processed by such a program, and
- the form of the output data resulting from the use of such a program.

3 Except where stated otherwise, requirements and prohibitions specified by this part of ISO/IEC 1539 apply to programs rather than processors.

4 This part of ISO/IEC 1539 does not specify

- the mechanism by which programs are transformed for use on computing systems,
- the operations required for setup and control of the use of programs on computing systems,
- the method of transcription of programs or their input or output data to or from a storage medium,
- the program and processor behavior when this part of ISO/IEC 1539 fails to establish an interpretation except for the processor detection and reporting requirements in items (2) to (8) of 1.5,
- the maximum number of images, or the size or complexity of a program and its data that will exceed the capacity of any particular computing system or the capability of a particular processor,
- the mechanism for determining the number of images of a program,
- the physical properties of an image or the relationship between images and the computational elements of a computing system,
- the physical properties of the representation of quantities and the method of rounding, approximating, or computing numeric values on a particular processor, except by reference to the IEEE International Standard under conditions specified in Clause 14,
- the physical properties of input/output records, files, and units, or
- the physical properties and implementation of storage.

1.2 Normative references

1 The following referenced standards are indispensable for the application 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/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

ISO/IEC 646:1991 (International Reference Version), *Information technology—ISO 7-bit coded character set for information interchange*

ISO/IEC 9899:1999, *Programming languages—C*

ISO/IEC 10646, *Information technology—Universal Multiple-Octet Coded Character Set (UCS)*

IEC 60559:1989, *Binary floating-point arithmetic for microprocessor systems*

1.3 Terms and definitions

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

1.3.1
abstract interface
set of procedure characteristics with dummy argument names (12.4.1)

1.3.2
actual argument
entity (R1223) that appears in a procedure reference

1.3.3
allocatable
having the ALLOCATABLE attribute (5.3.3)

1.3.4
array
set of scalar data, all of the same type and type parameters, whose individual elements are arranged in a rectangular pattern

1.3.4.1
array element
scalar individual element of an array

1.3.4.2
array pointer
array with the POINTER attribute (5.3.14)

1.3.4.3
array section
array subobject designated by *array-section*, and which is itself an array (6.5.3.3)

1.3.4.4
assumed-shape array
nonallocatable nonpointer dummy argument array that takes its shape from its effective argument (5.3.8.3)

1.3.4.5
assumed-size array
dummy argument array whose size is assumed from that of its effective argument (5.3.8.5)

1.3.4.6
deferred-shape array
allocatable array or array pointer, declared with a *deferred-shape-spec-list* (5.3.8.4)

1.3.4.7
explicit-shape array
array declared with an *explicit-shape-spec-list*, which specifies explicit values for the bounds in each dimension of the array (5.3.8.2)

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.5
ASCII character
character whose representation method corresponds to ISO/IEC 646:1991 (International Reference Version)

1.3.6
associate name
name of construct entity associated with a selector of an ASSOCIATE or SELECT TYPE construct (8.1.3)

1.3.7
associating entity
⟨in a dynamically-established association⟩ the entity that did not exist prior to the establishment of the association (16.5.5)

1.3.8
association
inheritance association (16.5.4), name association (16.5.1), pointer association (16.5.2), or storage association (16.5.3).

1.3.8.1
argument association
association between an effective argument and a dummy argument (12.5.2)

1.3.8.2
construct association
association between a selector and an associate name in an ASSOCIATE or SELECT TYPE construct (8.1.3, 8.1.9, 16.5.1.6)

1.3.8.3
host association
name association, other than argument association, between entities in a submodule or contained scoping unit and entities in its host (16.5.1.4)

1.3.8.4
inheritance association
association between the inherited components of an extended type and the components of its parent component

1.3.8.5
linkage association
association between a variable or common block with the BIND attribute and a C global variable (15.4, 16.5.1.5)

1.3.8.6
name association
argument association, construct association, host association, linkage association, or use association (16.5.1)

1.3.8.7
pointer association
association between a pointer and an entity with the TARGET attribute (16.5.2)

1.3.8.8
storage association
association between storage sequences (16.5.3)

1.3.8.9
use association
association between entities in a module and entities in a scoping unit or construct that references that module, as specified by a USE statement (11.2.2)

© ISO/IEC 2010 – All rights reserved

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.9
attribute
property of an entity that determines its uses (5.1)

1.3.10
automatic data object
automatic object
nondummy data object with a type parameter or array bound that depends on the value of a *specification-expr* that is not a constant expression

1.3.11
base object
⟨*data-ref*⟩ object designated by the leftmost *part-name* (6.4.2)

1.3.12
binding
type-bound procedure or final subroutine (4.5.5)

1.3.13
binding name
name given to a specific or generic type-bound procedure in the type definition (4.5.5)

1.3.14
binding label
default character value specifying the name by which a global entity with the BIND attribute is known to the companion processor (15.5.2, 15.4.2)

1.3.15
block
sequence of executable constructs formed by the syntactic class *block* and which is treated as a unit by the executable constructs described in 8.1

1.3.16
block data program unit
program unit whose initial statement is a BLOCK DATA statement, used for providing initial values for data objects in named common blocks (11.3)

1.3.17
bound
array bound
limit of a dimension of an array

1.3.18
branch target statement
action-stmt, *associate-stmt*, *end-associate-stmt*, *if-then-stmt*, *end-if-stmt*, *select-case-stmt*, *end-select-stmt*, *select-type-stmt*, *end-select-type-stmt*, *do-stmt*, *end-do-stmt*, *block-stmt*, *end-block-stmt*, *critical-stmt*, *end-critical-stmt*, a *forall-construct-stmt*, *do-term-action-stmt*, *do-term-shared-stmt*, or *where-construct-stmt* whose statement label appears as a *label* in a GO TO statement, computed GO TO statement, arithmetic IF statement, *alt-return-spec*, END= specifier, EOR= specifier, or ERR= specifier (8.2.1)

1.3.19
C address
value identifying the location of a data object or procedure either defined by the companion processor or which might be accessible to the companion processor

NOTE 1.1

This is the concept that ISO/IEC 9899:1999 calls the address.

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.20
character context
within a character literal constant (4.4.3) or within a character string edit descriptor (10.3.2)

1.3.21
characteristics
⟨dummy argument⟩ being a dummy data object, dummy procedure, or an asterisk (alternate return indicator)

1.3.22
characteristics
⟨dummy data object⟩ properties listed in 12.3.2.2

1.3.23
characteristics
⟨dummy procedure or dummy procedure pointer⟩ properties listed in 12.3.2.3

1.3.24
characteristics
⟨function result⟩ properties listed in 12.3.3

1.3.25
characteristics
⟨procedure⟩ properties listed in 12.3.1

1.3.26
coarray
data entity that has nonzero corank (2.4.7)

1.3.27
cobound
bound (limit) of a codimension

1.3.28
codimension
dimension of the pattern formed by a set of corresponding coarrays

1.3.29
coindexed object
data object whose designator includes an *image-selector* (R624, 6.6)

1.3.30
collating sequence
one-to-one mapping from a character set into the nonnegative integers (4.4.3.4)

1.3.31
common block
block of physical storage specified by a COMMON statement (5.7.2)

1.3.31.1
blank common
unnamed common block

1.3.32
companion processor
processor-dependent mechanism by which global data and procedures may be referenced or defined (2.5.7)

1.3.33
component
part of a derived type, or of an object of derived type, defined by a *component-def-stmt* (4.5.4)

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.33.1
direct component
one of the components, or one of the direct components of a nonpointer nonallocatable component (4.5.1)

1.3.33.2
parent component
component of an extended type whose type is that of the parent type and whose components are inheritance associated with the inherited components of the parent type (4.5.7.2)

1.3.33.3
subcomponent
⟨structure⟩ direct component that is a subobject of the structure (6.4.2)

1.3.33.4
ultimate component
component that is of intrinsic type, a pointer, or allocatable; or an ultimate component of a nonpointer nonallocatable component of derived type

1.3.34
component order
ordering of the nonparent components of a derived type that is used for intrinsic formatted input/output and structure constructors (where component keywords are not used) (4.5.4.7)

1.3.35
conformable
⟨of two data entities⟩ having the same shape, or one being an array and the other being scalar

1.3.36
connected
relationship between a unit and a file: each is connected if and only if the unit refers to the file (9.5.4)

1.3.37
constant
data object that has a value and which cannot be defined, redefined, or become undefined during execution of a program (3.2.3, 6.3)

1.3.37.1
literal constant
constant that does not have a name (R305, 4.4)

1.3.37.2
named constant
named data object with the PARAMETER attribute (5.3.13)

1.3.38
construct entity
entity whose identifier has the scope of a construct (16.1, 16.4)

1.3.39
constant expression
expression satisfying the requirements specified in 7.1.12, thus ensuring that its value is constant

1.3.40
contiguous
⟨array⟩ having array elements in order that are not separated by other data objects, as specified in 5.3.7

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.41
contiguous
⟨multi-part data object⟩ that the parts in order are not separated by other data objects

1.3.42
corank
number of codimensions of a coarray (zero for objects that are not coarrays)

1.3.43
cosubscript
(R625) scalar integer expression in an *image-selector* (R624)

1.3.44
data entity
data object, result of the evaluation of an expression, or the result of the execution of a function reference

1.3.45
data object
object
constant (4.1.3), variable (6), or subobject of a constant (2.4.3.2.3)

1.3.46
decimal symbol
character that separates the whole and fractional parts in the decimal representation of a real number in a file (10.6)

1.3.47
declaration
specification of attributes for various program entities

NOTE 1.2

Often this involves specifying the type of a named data object or specifying the shape of a named array object.

1.3.48
default initialization
mechanism for automatically initializing pointer components to have a defined pointer association status, and nonpointer components to have a particular value (4.5.4.6)

1.3.49
default-initialized
⟨subcomponent⟩ subject to a default initialization specified in the type definition for that component (4.5.4.6)

1.3.50
definable
capable of definition and permitted to become defined

1.3.51
defined
⟨data object⟩ has a valid value

1.3.52
defined
⟨pointer⟩ has a pointer association status of associated or disassociated (16.5.2.2)

1.3.53
defined assignment
assignment defined by a procedure (7.2.1.4, 12.4.3.4.3)

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.54
defined input/output
input/output defined by a procedure and accessed via a *defined-io-generic-spec* (R1208, 9.6.4.8)

1.3.55
defined operation
operation defined by a procedure (7.1.6.1, 12.4.3.4.2)

1.3.56
definition
⟨data object⟩ process by which the data object becomes defined (16.6.5)

1.3.57
definition
⟨derived type (4.5.2), enumeration (4.6), or procedure (12.6)⟩ specification of the type, enumeration, or procedure

1.3.58
descendant
⟨module or submodule⟩ submodule that extends that module or submodule or that extends another descendant thereof

1.3.59
designator
name followed by zero or more component selectors, complex part selectors, array section selectors, array element selectors, image selectors, and substring selectors (6.1)

1.3.59.1
complex part designator
designator that designates the real or imaginary part of a complex data object, independently of the other part (6.4.4)

1.3.59.2
object designator
data object designator
designator for a data object

NOTE 1.3

An object name is a special case of an object designator.

1.3.59.3
procedure designator
designator for a procedure

1.3.60
disassociated
⟨pointer association⟩ pointer association status of not being associated with any target and not being undefined (16.5.2.2)

1.3.61
disassociated
⟨pointer⟩ has a pointer association status of disassociated

1.3.62
dummy argument
entity whose identifier appears in a dummy argument list (R1235) in a FUNCTION, SUBROUTINE, ENTRY, or statement function statement, or whose name can be used as an argument keyword in a reference to an intrinsic procedure or a procedure in an intrinsic module

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.62.1
dummy data object
dummy argument that is a data object

1.3.62.2
dummy function
dummy procedure that is a function

1.3.63
effective argument
entity that is argument-associated with a dummy argument (12.5.2.3)

1.3.64
effective item
scalar object resulting from the application of the rules in 9.6.3 to an input/output list

1.3.65
elemental
independent scalar application of an action or operation to elements of an array or corresponding elements of a set of conformable arrays and scalars, or possessing the capability of elemental operation

NOTE 1.4

Combination of scalar and array operands or arguments combine the scalar operand(s) with each element of the array operand(s).

1.3.65.1
elemental assignment
assignment that operates elementally

1.3.65.2
elemental operation
operation that operates elementally

1.3.65.3
elemental operator
operator in an elemental operation

1.3.65.4
elemental procedure
elemental intrinsic procedure or procedure defined by an elemental subprogram

1.3.65.5
elemental reference
reference to an elemental procedure with at least one array actual argument

1.3.65.6
elemental subprogram
subprogram with the ELEMENTAL prefix

1.3.66
END statement
end-block-data-stmt, *end-function-stmt*, *end-module-stmt*, *end-mp-subprogram-stmt*, *end-program-stmt*, *end-submodule-stmt*, or *end-subroutine-stmt*

1.3.67
explicit initialization
initialization of a data object by a specification statement (5.2.3, 5.4.7)

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.68
explicit interface
interface of a procedure that includes all the characteristics of the procedure and names for its dummy arguments except for asterisk dummy arguments (12.4.2)

1.3.69
extent
number of elements in a single dimension of an array

1.3.70
external file
file that exists in a medium external to the program (9.3)

1.3.71
external unit
external input/output unit
entity that can be connected to an external file

1.3.72
file storage unit
unit of storage in a stream file or an unformatted record file (9.3.5)

1.3.73
final subroutine
subroutine whose name appears in a FINAL statement (4.5.6) in a type definition, and which can be automatically invoked by the processor when an object of that type is finalized (4.5.6.2)

1.3.74
finalizable
⟨type⟩ has a final subroutine or a nonpointer nonallocatable component of finalizable type

1.3.75
finalizable
⟨nonpointer data entity⟩ of finalizable type

1.3.76
finalization
process of calling final subroutines when one of the events listed in 4.5.6.3 occurs

1.3.77
function
procedure that is invoked by an expression

1.3.78
generic identifier
lexical token that identifies a generic set of procedures, intrinsic operations, and/or intrinsic assignments

1.3.79
host instance
⟨internal procedure, or dummy procedure or procedure pointer associated with an internal procedure⟩ instance of the host procedure that supplies the host environment of the internal procedure (12.6.2.4)

1.3.80
host scoping unit
host
scoping unit immediately surrounding another scoping unit, or the scoping unit extended by a submodule

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.81
IEEE infinity
IEC 60559:1989 conformant infinite floating-point value

1.3.82
IEEE NaN
IEC 60559:1989 conformant floating-point datum that does not represent a number

1.3.83
image
instance of a Fortran program (2.3.4)

1.3.84
image index
integer value identifying an image

1.3.85
image control statement
statement that affects the execution ordering between images (8.5)

1.3.86
implicit interface
interface of a procedure that includes only the type and type parameters of a function result (12.4.2, 12.4.3.8)

1.3.87
inclusive scope
nonblock scoping unit plus every block scoping unit whose host is that scoping unit or that is nested within such a block scoping unit

NOTE 1.5

That is, inclusive scope is the scope as if BLOCK constructs were not scoping units.

1.3.88
inherit
⟨extended type⟩ acquire entities (components, type-bound procedures, and type parameters) through type extension from the parent type

1.3.89
inquiry function
intrinsic function, or function in an intrinsic module, whose result depends on the properties of one or more of its arguments instead of their values

1.3.90
interface
⟨procedure⟩ name, procedure characteristics, dummy argument names, binding label, and generic identifiers (12.4.1)

1.3.90.1
generic interface
set of procedure interfaces identified by a generic identifier

1.3.90.2
specific interface
interface identified by a nongeneric name

1.3.91
interface block
abstract interface block, generic interface block, or specific interface block (12.4.3.2)

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.91.1
abstract interface block
interface block with the ABSTRACT keyword; collection of interface bodies that specify named abstract interfaces

1.3.91.2
generic interface block
interface block with a *generic-spec*; collection of interface bodies and procedure statements that are to be given that generic identifier

1.3.91.3
specific interface block
interface block with no *generic-spec* or ABSTRACT keyword; collection of interface bodies that specify the interfaces of procedures

1.3.92
interoperable
⟨Fortran entity⟩ equivalent to an entity defined by or definable by the companion processor (15.3)

1.3.93
intrinsic
type, procedure, module, assignment, operator, or input/output operation defined in this part of ISO/IEC 1539 and accessible without further definition or specification, or a procedure or module provided by a processor but not defined in this part of ISO/IEC 1539

1.3.93.1
standard intrinsic
⟨procedure or module⟩ defined in this part of ISO/IEC 1539 (13)

1.3.93.2
nonstandard intrinsic
⟨procedure or module⟩ provided by a processor but not defined in this part of ISO/IEC 1539

1.3.94
internal file
character variable that is connected to an internal unit (9.4)

1.3.95
internal unit
input/output unit that is connected to an internal file (9.5.4)

1.3.96
ISO 10646 character
character whose representation method corresponds to UCS-4 in ISO/IEC 10646

1.3.97
keyword
statement keyword, argument keyword, type parameter keyword, or component keyword

1.3.97.1
argument keyword
word that identifies the corresponding dummy argument in an actual argument list

1.3.97.2
component keyword
word that identifies a component in a structure constructor

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.97.3
statement keyword
word that is part of the syntax of a statement (2.5.2)

1.3.97.4
type parameter keyword
word that identifies a type parameter in a type parameter list

1.3.98
lexical token
keyword, name, literal constant other than a complex literal constant, operator, label, delimiter, comma, =, =>, :, ::, ;, or % (3.2)

1.3.99
line
sequence of zero or more characters

1.3.100
main program
program unit that is not a subprogram, module, submodule, or block data program unit (11.1)

1.3.101
masked array assignment
assignment statement in a WHERE statement or WHERE construct (7.2.3)

1.3.102
module
program unit containing (or accessing from other modules) definitions that are to be made accessible to other program units (11.2)

1.3.103
name
identifier of a program consituent, formed according to the rules given in 3.2.2

1.3.104
NaN
Not a Number, a symbolic floating-point datum (IEC 60559:1989)

1.3.105
operand
data value that is the subject of an operator

1.3.106
operator
intrinsic-operator, *defined-unary-op*, or *defined-binary-op* (R309, R703, R723)

1.3.107
passed-object dummy argument
dummy argument of a type-bound procedure or procedure pointer component that becomes associated with the object through which the procedure is invoked (4.5.4.5)

1.3.108
pointer
data pointer (1.3) or procedure pointer (1.3)

1.3.108.1
data pointer
data entity with the POINTER attribute (5.3.14)

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.108.2
procedure pointer
procedure with the EXTERNAL and POINTER attributes (5.3.9, 5.3.14)

1.3.109
pointer assignment
association of a pointer with a target, by execution of a pointer assignment statement (7.2.2) or an intrinsic assignment statement (7.2.1.2) for a derived-type object that has the pointer as a subobject

1.3.110
polymorphic
⟨data entity⟩ able to be of differing dynamic types during program execution (4.3.1.3)

1.3.111
preconnected
⟨file or unit⟩ connected at the beginning of execution of the program (9.5.5)

1.3.112
procedure
entity encapsulating an arbitrary sequence of actions that can be invoked directly during program execution

1.3.112.1
dummy procedure
procedure that is a dummy argument (12.2.2.3)

1.3.112.2
external procedure
procedure defined by an external subprogram (R203) or by means other than Fortran (12.6.3)

1.3.112.3
internal procedure
procedure defined by an internal subprogram (R211)

1.3.112.4
module procedure
procedure that is defined by a module subprogram (R1108)

1.3.112.5
pure procedure
procedure declared or defined to be pure according to the rules in 12.7

1.3.112.6
type-bound procedure
procedure that is bound to a derived type and referenced via an object of that type (4.5.5)

1.3.113
processor
combination of a computing system and mechanism by which programs are transformed for use on that computing system

1.3.114
processor dependent
not completely specified in this part of ISO/IEC 1539, having methods and semantics determined by the processor

1.3.115
program
set of Fortran program units and entities defined by means other than Fortran that includes exactly one main program

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.116
program unit
main program, external subprogram, module, submodule, or block data program unit (2.2.1)

1.3.117
rank
number of array dimensions of a data entity (zero for a scalar entity)

1.3.118
record
sequence of values or characters in a file (9.2)

1.3.119
record file
file composed of a sequence of records (9.1)

1.3.120
reference
data object reference, procedure reference, or module reference

1.3.120.1
data object reference
appearance of a data object designator (6.1) in a context requiring its value at that point during execution

1.3.120.2
function reference
appearance of the procedure designator for a function, or operator symbol in a context requiring execution of the function during expression evaluation (12.5.3)

1.3.120.3
module reference
appearance of a module name in a USE statement (11.2.2)

1.3.120.4
procedure reference
appearance of a procedure designator, operator symbol, or assignment symbol in a context requiring execution of the procedure at that point during execution; or occurrence of defined input/output (10.7.6) or derived-type finalization (4.5.6.2)

1.3.121
result variable
variable that returns the value of a function (12.6.2.2)

1.3.122
saved
having the SAVE attribute (5.3.16)

1.3.123
scalar
data entity that can be represented by a single value of the type and that is not an array (6.5)

1.3.124
scoping unit
BLOCK construct, derived-type definition, interface body, program unit, or subprogram, excluding all nested scoping units in it

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.124.1
block scoping unit
scoping unit of a BLOCK construct

1.3.125
sequence
set of elements ordered by a one-to-one correspondence with the numbers 1, 2, to n

1.3.126
sequence structure
scalar data object of a sequence type (4.5.2.3)

1.3.127
sequence type
derived type with the SEQUENCE attribute (4.5.2.3)

1.3.127.1
character sequence type
sequence type with no type parameters, no allocatable or pointer components, and whose components are all default character or of another character sequence type

1.3.127.2
numeric sequence type
sequence type with no type parameters, no allocatable or pointer components, and whose components are all default complex, default integer, default logical, default real, double precision real, or of another numeric sequence type

1.3.128
shape
array dimensionality of a data entity, represented as a rank-one array whose size is the rank of the data entity and whose elements are the extents of the data entity

NOTE 1.6

Thus the shape of a scalar data entity is an array with rank one and size zero.

1.3.129
simply contiguous
⟨array designator or variable⟩ satisfying the conditions specified in 6.5.4

NOTE 1.7

These conditions are simple ones which make it clear that the designator or variable designates a contiguous array.

1.3.130
size
⟨array⟩ total number of elements in the array

1.3.131
specification expression
expression satisfying the requirements specified in 7.1.11, thus being suitable for use in specifications

1.3.132
specific name
name that is not a generic name

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.133
standard-conforming program
program that uses only those forms and relationships described in, and has an interpretation according to, this part of ISO/IEC 1539

1.3.134
statement
sequence of one or more complete or partial lines satisfying a syntax rule that ends in *-stmt* (3.3)

1.3.134.1
executable statement
statement that is a member of the syntactic class *executable-construct*, excluding those in the *specification-part* of a BLOCK construct

1.3.134.2
nonexecutable statement
statement that is not an executable statement

1.3.135
statement entity
entity whose identifier has the scope of a statement or part of a statement (16.1, 16.4)

1.3.136
statement label
label
unsigned positive number of up to five digits that refers to an individual statement (3.2.5)

1.3.137
storage sequence
contiguous sequence of storage units (16.5.3.2)

1.3.138
storage unit
character storage unit, numeric storage unit, file storage unit, or unspecified storage unit (16.5.3.2)

1.3.138.1
character storage unit
unit of storage that holds a default character value (16.5.3.2)

1.3.138.2
numeric storage unit
unit of storage that holds a default real, default integer, or default logical value (16.5.3.2)

1.3.138.3
unspecified storage unit
unit of storage that holds a value that is not default character, default real, double precision real, default logical, or default complex (16.5.3.2)

1.3.139
stream file
file composed of a sequence of file storage units (9.1)

1.3.140
structure
scalar data object of derived type (4.5)

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.140.1
structure component
component of a structure

1.3.140.2
structure constructor
syntax (*structure-constructor*, 4.5.10) that specifies a structure value or creates such a value

1.3.141
submodule
program unit that extends a module or another submodule (11.2.3)

1.3.142
subobject
portion of data object that can be referenced, and if it is a variable defined, independently of any other portion

1.3.143
subprogram
function-subprogram (R1227) or *subroutine-subprogram* (R1233)

1.3.143.1
external subprogram
subprogram that is not contained in a main program, module, submodule, or another subprogram

1.3.143.2
internal subprogram
subprogram that is contained in a main program or another subprogram

1.3.143.3
module subprogram
subprogram that is contained in a module or submodule but is not an internal subprogram

1.3.144
subroutine
procedure invoked by a CALL statement, by defined assignment, or by some operations on derived-type entities

1.3.144.1
atomic subroutine
intrinsic subroutine that performs an action on its ATOM argument atomically

1.3.145
target
entity that is pointer associated with a pointer (16.5.2.2), entity on the right-hand-side of a pointer assignment statement (R733), or entity with the TARGET attribute (5.3.17)

1.3.146
transformational function
intrinsic function, or function in an intrinsic module, that is neither elemental nor an inquiry function

1.3.147
type
data type
named category of data characterized by a set of values, a syntax for denoting these values, and a set of operations that interpret and manipulate the values (4.1)

1.3.147.1
abstract type
type with the ABSTRACT attribute (4.5.7.1)

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.147.2
declared type
type that a data entity is declared to have, either explicitly or implicitly (4.3.1, 7.1.9)

1.3.147.3
derived type
type defined by a type definition (4.5) or by an intrinsic module

1.3.147.4
dynamic type
type of a data entity at a particular point during execution of a program (4.3.1.3, 7.1.9)

1.3.147.5
extended type
type with the EXTENDS attribute (4.5.7.1)

1.3.147.6
extensible type
type that has neither the BIND attribute nor the SEQUENCE attribute and which therefore may be extended using the EXTENDS clause (4.5.7.1)

1.3.147.7
extension type
⟨of one type with respect to another⟩ is the same type or is an extended type whose parent type is an extension type of the other type

1.3.147.8
intrinsic type
type defined by this part of ISO/IEC 1539 that is always accessible (4.4)

1.3.147.9
numeric type
one of the types integer, real, and complex

1.3.147.10
parent type
⟨extended type⟩ type named in the EXTENDS clause

1.3.147.11
type compatible
compatibility of the type of one entity with respect to another for purposes such as argument association, pointer association, and allocation (4.3.1)

1.3.147.12
type parameter
value used to parameterize a type (4.2)

1.3.147.12.1
assumed type parameter
length type parameter that assumes the type parameter value from another entity

NOTE 1.8

The other entity is

- the selector for an associate name,
- the *constant-expr* for a named constant of type character, or
- the effective argument for a dummy argument.

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1.3.147.12.2
deferred type parameter
length type parameter whose value can change during execution of a program and whose *type-param-value* is a colon

1.3.147.12.3
kind type parameter
type parameter whose value is required to be defaulted or given by a constant expression

1.3.147.12.4
length type parameter
type parameter whose value is permitted to be assumed, deferred, or given by a specification expression

1.3.147.12.5
type parameter inquiry
syntax (*type-param-inquiry*) that is used to inquire the value of a type parameter of a data object (6.4.5)

1.3.147.12.6
type parameter order
ordering of the type parameters of a type (4.5.3.2) used for derived-type specifiers (*derived-type-spec*, 4.5.9)

1.3.148
ultimate argument
nondummy entity with which a dummy argument is associated via a chain of argument associations (12.5.2.3)

1.3.149
undefined
⟨data object⟩ does not have a valid value

1.3.150
undefined
⟨pointer⟩ does not have a pointer association status of associated or disassociated (16.5.2.2)

1.3.151
unit
input/output unit
means, specified by an *io-unit*, for referring to a file (9.5.1)

1.3.152
unlimited polymorphic
able to have any dynamic type during program execution (4.3.1.3)

1.3.153
unsaved
not having the SAVE attribute (5.3.16)

1.3.154
variable
data entity that can be defined and redefined during execution of a program

1.3.154.1
local variable
variable in a scoping unit that is not a dummy argument or part thereof, is not a global entity or part thereof, and is not accessible outside that scoping unit

1.3.154.2
lock variable
scalar variable of type LOCK_TYPE (13.8.2.16) from the intrinsic module ISO_FORTRAN_ENV

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.3.155
vector subscript
section-subscript that is an array (6.5.3.3.2)

1.3.156
whole array
array component or array name without further qualification (6.5.2)

1.4 Notation, symbols and abbreviated terms

1.4.1 Syntax rules

1 Syntax rules describe the forms that Fortran lexical tokens, statements, and constructs may take. These syntax rules are expressed in a variation of Backus-Naur form (BNF) with the following conventions.

- Characters from the Fortran character set (3.1) are interpreted literally as shown, except where otherwise noted.
- Lower-case italicized letters and words (often hyphenated and abbreviated) represent general syntactic classes for which particular syntactic entities shall be substituted in actual statements.

  Common abbreviations used in syntactic terms are:

arg	for	argument	*attr*	for	attribute
decl	for	declaration	*def*	for	definition
desc	for	descriptor	*expr*	for	expression
int	for	integer	*op*	for	operator
spec	for	specifier	*stmt*	for	statement

- The syntactic metasymbols used are:

is	introduces a syntactic class definition
or	introduces a syntactic class alternative
[ ]	encloses an optional item
[ ] ...	encloses an optionally repeated item that may occur zero or more times
■	continues a syntax rule

- Each syntax rule is given a unique identifying number of the form Rsnn, where s is a one- or two-digit clause number and nn is a two-digit sequence number within that clause. The syntax rules are distributed as appropriate throughout the text, and are referenced by number as needed. Some rules in Clauses 2 and 3 are more fully described in later clauses; in such cases, the clause number s is the number of the later clause where the rule is repeated.
- The syntax rules are not a complete and accurate syntax description of Fortran, and cannot be used to generate a Fortran parser automatically; where a syntax rule is incomplete, it is restricted by corresponding constraints and text.

NOTE 1.9

An example of the use of the syntax rules is:

digit-string **is** *digit* [*digit*] ...

The following are examples of forms for a digit string allowed by the above rule:

digit
digit digit
digit digit digit digit

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

NOTE 1.9 (cont.)

digit digit digit digit digit digit digit digit

If particular entities are substituted for *digit*, actual digit strings might be:

```
4
67
1999
10243852
```

1.4.2 Constraints

1 Each constraint is given a unique identifying number of the form Csnn, where s is a one or two digit clause number and nn is a two or three digit sequence number within that clause.

2 Often a constraint is associated with a particular syntax rule. Where that is the case, the constraint is annotated with the syntax rule number in parentheses. A constraint that is associated with a syntax rule constitutes part of the definition of the syntax term defined by the rule. It thus applies in all places where the syntax term appears.

3 Some constraints are not associated with particular syntax rules. The effect of such a constraint is similar to that of a restriction stated in the text, except that a processor is required to have the capability to detect and report violations of constraints (1.5). In some cases, a broad requirement is stated in text and a subset of the same requirement is also stated as a constraint. This indicates that a standard-conforming program is required to adhere to the broad requirement, but that a standard-conforming processor is required only to have the capability of diagnosing violations of the constraint.

1.4.3 Assumed syntax rules

1 In order to minimize the number of additional syntax rules and convey appropriate constraint information, the following rules are assumed.

R101 *xyz-list* **is** *xyz* [, *xyz*] ...

R102 *xyz-name* **is** *name*

R103 *scalar-xyz* **is** *xyz*

C101 (R103) *scalar-xyz* shall be scalar.

2 The letters "*xyz*" stand for any syntactic class phrase. An explicit syntax rule for a term overrides an assumed rule.

1.4.4 Syntax conventions and characteristics

1 Any syntactic class name ending in "-*stmt*" follows the source form statement rules: it shall be delimited by end-of-line or semicolon, and may be labeled unless it forms part of another statement (such as an IF or WHERE statement). Conversely, everything considered to be a source form statement is given a "-*stmt*" ending in the syntax rules.

2 The rules on statement ordering are described rigorously in the definition of *program-unit* (R202). Expression hierarchy is described rigorously in the definition of *expr* (R722).

3 The suffix "-*spec*" is used consistently for specifiers, such as input/output statement specifiers. It also is used for type declaration attribute specifications (for example, "*array-spec*" in R515), and in a few other cases.

4 Where reference is made to a type parameter, including the surrounding parentheses, the suffix "-*selector*" is used. See, for example, "*kind-selector*" (R405) and "*length-selector*" (R421).

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.4.5 Text conventions

1 In descriptive text, an equivalent English word is frequently used in place of a syntactic term. Particular statements and attributes are identified in the text by an upper-case keyword, e.g., "END statement". The descriptions of obsolescent features appear in a smaller type size.

NOTE 1.10

This sentence is an example of the type size used for obsolescent features.

1.5 Conformance

1 A program (2.2.2) is a standard-conforming program if it uses only those forms and relationships described herein and if the program has an interpretation according to this part of ISO/IEC 1539. A program unit (2.2.1) conforms to this part of ISO/IEC 1539 if it can be included in a program in a manner that allows the program to be standard conforming.

2 A processor conforms to this part of ISO/IEC 1539 if:

(1) it executes any standard-conforming program in a manner that fulfills the interpretations herein, subject to any limits that the processor may impose on the size and complexity of the program;

(2) it contains the capability to detect and report the use within a submitted program unit of a form designated herein as obsolescent, insofar as such use can be detected by reference to the numbered syntax rules and constraints;

(3) it contains the capability to detect and report the use within a submitted program unit of an additional form or relationship that is not permitted by the numbered syntax rules or constraints, including the deleted features described in Annex B

(4) it contains the capability to detect and report the use within a submitted program unit of an intrinsic type with a kind type parameter value not supported by the processor (4.4);

(5) it contains the capability to detect and report the use within a submitted program unit of source form or characters not permitted by Clause 3;

(6) it contains the capability to detect and report the use within a submitted program of name usage not consistent with the scope rules for names, labels, operators, and assignment symbols in Clause 16;

(7) it contains the capability to detect and report the use within a submitted program unit of intrinsic procedures whose names are not defined in Clause 13; and

(8) it contains the capability to detect and report the reason for rejecting a submitted program.

3 However, in a format specification that is not part of a FORMAT statement (10.2.1), a processor need not detect or report the use of deleted or obsolescent features, or the use of additional forms or relationships.

4 A standard-conforming processor may allow additional forms and relationships provided that such additions do not conflict with the standard forms and relationships. However, a standard-conforming processor may allow additional intrinsic procedures even though this could cause a conflict with the name of a procedure in a standard-conforming program. If such a conflict occurs and involves the name of an external procedure, the processor is permitted to use the intrinsic procedure unless the name is given the EXTERNAL attribute (5.3.9) in its scope (2.2.1). A standard-conforming program shall not use nonstandard intrinsic procedures or modules that have been added by the processor.

5 Because a standard-conforming program may place demands on a processor that are not within the scope of this part of ISO/IEC 1539 or may include standard items that are not portable, such as external procedures defined by means other than Fortran, conformance to this part of ISO/IEC 1539 does not ensure that a program will execute consistently on all or any standard-conforming processors.

6 The semantics of facilities that are identified as processor dependent are not completely specified in this part of ISO/IEC 1539. They shall be provided, with methods or semantics determined by the processor.

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

NOTE 1.11

The processor should be accompanied by documentation that specifies the limits it imposes on the size and complexity of a program and the means of reporting when these limits are exceeded, that defines the additional forms and relationships it allows, and that defines the means of reporting the use of additional forms and relationships and the use of deleted or obsolescent forms. In this context, the use of a deleted form is the use of an additional form.

The processor should be accompanied by documentation that specifies the methods or semantics of processor-dependent facilities.

1.6 Compatibility

1.6.1 New intrinsic procedures

1 Each Fortran International Standard since ISO 1539:1980 (informally referred to as FORTRAN 77), defines more intrinsic procedures than the previous one. Therefore, a Fortran program conforming to an older standard might have a different interpretation under a newer standard if it invokes an external procedure having the same name as one of the new standard intrinsic procedures, unless that procedure is specified to have the EXTERNAL attribute.

1.6.2 Fortran 2003 compatibility

1 This part of ISO/IEC 1539 is an upward compatible extension to the preceding Fortran International Standard, ISO/IEC 1539-1:2004 (Fortran 2003). Any standard-conforming Fortran 2003 program remains standard-conforming under this part of ISO/IEC 1539.

1.6.3 Fortran 95 compatibility

1 Except as identified in this subclause, this part of ISO/IEC 1539 is an upward compatible extension to ISO/IEC 1539-1:1997 (Fortran 95). Any standard-conforming Fortran 95 program remains standard-conforming under this part of ISO/IEC 1539. The following Fortran 95 features might have different interpretations in this part of ISO/IEC 1539.

- Earlier Fortran standards had the concept of printing, meaning that column one of formatted output had special meaning for a processor-dependent (possibly empty) set of external files. This could be neither detected nor specified by a standard-specified means. The interpretation of the first column is not specified by this part of ISO/IEC 1539.
- This part of ISO/IEC 1539 specifies a different output format for real zero values in list-directed and namelist output.
- If the processor can distinguish between positive and negative real zero, this part of ISO/IEC 1539 requires different returned values for ATAN2(Y,X) when X $<$ 0 and Y is negative real zero and for LOG(X) and SQRT(X) when X is complex with REAL(X) $<$ 0 and negative zero imaginary part.
- This part of ISO/IEC 1539 has fewer restrictions on constant expressions than Fortran 95; this might affect whether a variable is considered to be automatic.

1.6.4 Fortran 90 compatibility

1 Except for the deleted features noted in Annex B.1, and except as identified in this subclause, this part of ISO/IEC 1539 is an upward compatible extension to ISO/IEC 1539:1991 (Fortran 90). Any standard-conforming Fortran 90 program that does not use one of the deleted features remains standard-conforming under this part of ISO/IEC 1539.

2 The PAD= specifier in the INQUIRE statement in this part of ISO/IEC 1539 returns the value UNDEFINED if there is no connection or the connection is for unformatted input/output. Fortran 90 specified YES.

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

3 Fortran 90 specified that if the second argument to MOD or MODULO was zero, the result was processor dependent. This part of ISO/IEC 1539 specifies that the second argument shall not be zero.

4 The following Fortran 90 features have different interpretations in this part of ISO/IEC 1539:

- the result value of the intrinsic function SIGN (when the second argument is a negative real zero);
- formatted output of negative real values (when the output value is zero);
- whether an expression is a constant expression (thus whether a variable is considered to be automatic).

1.6.5 Fortran 77 compatibility

1 Except for the deleted features noted in Annex B.1, and except as identified in this subclause, this part of ISO/IEC 1539 is an upward compatible extension to ISO 1539:1980 (Fortran 77). Any standard-conforming Fortran 77 program that does not use one of the deleted features noted in Annex B.1 and that does not depend on the differences specified here remains standard-conforming under this part of ISO/IEC 1539. This part of ISO/IEC 1539 restricts the behavior for some features that were processor dependent in Fortran 77. Therefore, a standard-conforming Fortran 77 program that uses one of these processor-dependent features might have a different interpretation under this part of ISO/IEC 1539, yet remain a standard-conforming program. The following Fortran 77 features might have different interpretations in this part of ISO/IEC 1539.

- Fortran 77 permitted a processor to supply more precision derived from a default real constant than can be represented in a default real datum when the constant is used to initialize a double precision real data object in a DATA statement. This part of ISO/IEC 1539 does not permit a processor this option.
- If a named variable that was not in a common block was initialized in a DATA statement and did not have the SAVE attribute specified, Fortran 77 left its SAVE attribute processor dependent. This part of ISO/IEC 1539 specifies (5.4.7) that this named variable has the SAVE attribute.
- Fortran 77 specified that the number of characters required by the input list was to be less than or equal to the number of characters in the record during formatted input. This part of ISO/IEC 1539 specifies (9.6.4.5.3) that the input record is logically padded with blanks if there are not enough characters in the record, unless the PAD= specifier with the value 'NO' is specified in an appropriate OPEN or READ statement.
- A value of 0 for a list item in a formatted output statement will be formatted in a different form for some G edit descriptors. In addition, this part of ISO/IEC 1539 specifies how rounding of values will affect the output field form, but Fortran 77 did not address this issue. Therefore, some Fortran 77 processors might produce an output form different from the output form produced by Fortran 2003 processors for certain combinations of values and G edit descriptors.
- If the processor can distinguish between positive and negative real zero, the behavior of the intrinsic function SIGN when the second argument is negative real zero is changed by this part of ISO/IEC 1539.

1.7 Deleted and obsolescent features

1.7.1 General

1 This part of ISO/IEC 1539 protects the users' investment in existing software by including all but six of the language elements of Fortran 90 that are not processor dependent. This part of ISO/IEC 1539 identifies two categories of outmoded features. The first category, deleted features, consists of features considered to have been redundant in Fortran 77 and largely unused in Fortran 90. Those in the second category, obsolescent features, are considered to have been redundant in Fortran 90 and Fortran 95, but are still frequently used.

1.7.2 Nature of deleted features

1 Better methods existed in Fortran 77 for each deleted feature. These features were not included in Fortran 95 or Fortran 2003, and are not included in this revision of Fortran.

© ISO/IEC 2010 – All rights reserved

AVA Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

1.7.3 Nature of obsolescent features

1 Better methods existed in Fortran 90 and Fortran 95 for each obsolescent feature. It is recommended that programmers use these better methods in new programs and convert existing code to these methods.

2 The obsolescent features are identified in the text of this part of ISO/IEC 1539 by a distinguishing type font (1.4.5).

3 A future revision of this part of ISO/IEC 1539 might delete an obsolescent feature if its use has become insignificant.

© ISO/IEC 2010 – All rights reserved
Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

2 Fortran concepts

2.1 High level syntax

1 This subclause introduces the syntax associated with program units and other Fortran concepts above the construct, statement, and expression levels and illustrates their relationships.

NOTE 2.1

Constraints and other information related to the rules that do not begin with R2 appear in the appropriate clause.

R201 *program* **is** *program-unit*
[*program-unit*] ...

R202 *program-unit* **is** *main-program*
or *external-subprogram*
or *module*
or *submodule*
or *block-data*

R1101 *main-program* **is** [*program-stmt*]
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-program-stmt

R203 *external-subprogram* **is** *function-subprogram*
or *subroutine-subprogram*

R1227 *function-subprogram* **is** *function-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-function-stmt

R1233 *subroutine-subprogram* **is** *subroutine-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-subroutine-stmt

R1104 *module* **is** *module-stmt*
[*specification-part*]
[*module-subprogram-part*]
end-module-stmt

R1116 *submodule* **is** *submodule-stmt*
[*specification-part*]
[*module-subprogram-part*]
end-submodule-stmt

R1120 *block-data* **is** *block-data-stmt*
[*specification-part*]
end-block-data-stmt

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

R204 *specification-part* **is** [*use-stmt*] ...
[*import-stmt*] ...
[*implicit-part*]
[*declaration-construct*] ...

R205 *implicit-part* **is** [*implicit-part-stmt*] ...
implicit-stmt

R206 *implicit-part-stmt* **is** *implicit-stmt*
or *parameter-stmt*
or *format-stmt*
or *entry-stmt*

R207 *declaration-construct* **is** *derived-type-def*
or *entry-stmt*
or *enum-def*
or *format-stmt*
or *interface-block*
or *parameter-stmt*
or *procedure-declaration-stmt*
or *other-specification-stmt*
or *type-declaration-stmt*
or *stmt-function-stmt*

R208 *execution-part* **is** *executable-construct*
[*execution-part-construct*] ...

R209 *execution-part-construct* **is** *executable-construct*
or *format-stmt*
or *entry-stmt*
or *data-stmt*

R210 *internal-subprogram-part* **is** *contains-stmt*
[*internal-subprogram*] ...

R211 *internal-subprogram* **is** *function-subprogram*
or *subroutine-subprogram*

R1107 *module-subprogram-part* **is** *contains-stmt*
[*module-subprogram*] ...

R1108 *module-subprogram* **is** *function-subprogram*
or *subroutine-subprogram*
or *separate-module-subprogram*

R1237 *separate-module-subprogram* **is** *mp-subprogram-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-mp-subprogram-stmt

R212 *other-specification-stmt* **is** *access-stmt*
or *allocatable-stmt*
or *asynchronous-stmt*
or *bind-stmt*
or *codimension-stmt*
or *common-stmt*
or *data-stmt*

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

		or	*dimension-stmt*
		or	*equivalence-stmt*
		or	*external-stmt*
		or	*intent-stmt*
		or	*intrinsic-stmt*
		or	*namelist-stmt*
		or	*optional-stmt*
		or	*pointer-stmt*
		or	*protected-stmt*
		or	*save-stmt*
		or	*target-stmt*
		or	*volatile-stmt*
		or	*value-stmt*
R213	*executable-construct*	**is**	*action-stmt*
		or	*associate-construct*
		or	*block-construct*
		or	*case-construct*
		or	*critical-construct*
		or	*do-construct*
		or	*forall-construct*
		or	*if-construct*
		or	*select-type-construct*
		or	*where-construct*
R214	*action-stmt*	**is**	*allocate-stmt*
		or	*assignment-stmt*
		or	*backspace-stmt*
		or	*call-stmt*
		or	*close-stmt*
		or	*continue-stmt*
		or	*cycle-stmt*
		or	*deallocate-stmt*
		or	*end-function-stmt*
		or	*end-mp-subprogram-stmt*
		or	*end-program-stmt*
		or	*end-subroutine-stmt*
		or	*endfile-stmt*
		or	*error-stop-stmt*
		or	*exit-stmt*
		or	*flush-stmt*
		or	*forall-stmt*
		or	*goto-stmt*
		or	*if-stmt*
		or	*inquire-stmt*
		or	*lock-stmt*
		or	*nullify-stmt*
		or	*open-stmt*
		or	*pointer-assignment-stmt*
		or	*print-stmt*
		or	*read-stmt*
		or	*return-stmt*
		or	*rewind-stmt*
		or	*stop-stmt*
		or	*sync-all-stmt*
		or	*sync-images-stmt*

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010

or *sync-memory-stmt*
or *unlock-stmt*
or *wait-stmt*
or *where-stmt*
or *write-stmt*
or *arithmetic-if-stmt*
or *computed-goto-stmt*

C201 (R208) An *execution-part* shall not contain an *end-function-stmt*, *end-mp-subprogram-stmt*, *end-program-stmt*, or *end-subroutine-stmt*.

2.2 Program unit concepts

2.2.1 Program units and scoping units

1 Program units are the fundamental components of a Fortran program. A program unit is a main program, an external subprogram, a module, a submodule, or a block data program unit.

2 A subprogram is a function subprogram or a subroutine subprogram. A module contains definitions that are to be made accessible to other program units. A submodule is an extension of a module; it may contain the definitions of procedures declared in a module or another submodule. A block data program unit is used to specify initial values for data objects in named common blocks.

3 Each type of program unit is described in Clause 11 or 12.

4 A program unit consists of a set of nonoverlapping scoping units.

NOTE 2.2

The module or submodule containing a module subprogram is the host scoping unit of the module subprogram. The containing main program or subprogram is the host scoping unit of an internal subprogram.

An internal procedure is local to its host in the sense that its name is accessible within the host scoping unit and all its other internal procedures but is not accessible elsewhere.

2.2.2 Program

1 A program shall consist of exactly one main program, any number (including zero) of other kinds of program units, any number (including zero) of external procedures, and any number (including zero) of other entities defined by means other than Fortran. The main program shall be defined by a Fortran *main-program program-unit* or by means other than Fortran, but not both.

NOTE 2.3

There is a restriction that there shall be no more than one unnamed block data program unit (11.3).

2.2.3 Procedure

1 A procedure is either a function or a subroutine. Invocation of a function in an expression causes a value to be computed which is then used in evaluating the expression.

2 A procedure that is not pure might change the program state by changing the value of data objects accessible to it.

3 Procedures are described further in Clause 12.

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

2.2.4 Module

1 A module contains (or accesses from other modules) definitions that are to be made accessible to other program units. These definitions include data object declarations, type definitions, procedure definitions, and interface blocks. A scoping unit in another program unit may access the definitions in a module. Modules are further described in Clause 11.

2.2.5 Submodule

1 A submodule extends a module or another submodule.

2 It may provide definitions (12.6) for procedures whose interfaces are declared (12.4.3.2) in an ancestor module or submodule. It may also contain declarations and definitions of other entities, which are accessible in its descendants. An entity declared in a submodule is not accessible by use association unless it is a module procedure whose interface is declared in the ancestor module. Submodules are further described in Clause 11.

NOTE 2.4

A submodule has access to entities in its parent module or submodule by host association.

2.3 Execution concepts

2.3.1 Statement classification

1 Each Fortran statement is classified as either an executable statement or a nonexecutable statement.

2 An executable statement is an instruction to perform or control an action. Thus, the executable statements of a program unit determine the behavior of the program unit.

3 Nonexecutable statements are used to configure the program environment in which actions take place.

2.3.2 Statement order

Table 2.1: **Requirements on statement ordering**

<table>
<tr><td colspan="3">PROGRAM, FUNCTION, SUBROUTINE,
MODULE, SUBMODULE, or BLOCK DATA statement</td></tr>
<tr><td colspan="3">USE statements</td></tr>
<tr><td colspan="3">IMPORT statements</td></tr>
<tr><td rowspan="4">FORMAT
and
ENTRY
statements</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>PARAMETER
statements</td><td>IMPLICIT
statements</td></tr>
<tr><td>PARAMETER
and DATA
statements</td><td>Derived-type definitions,
interface blocks,
type declaration statements,
enumeration definitions,
procedure declarations,
specification statements,
and statement function statements</td></tr>
<tr><td>DATA
statements</td><td>Executable
constructs</td></tr>
<tr><td colspan="3">CONTAINS statement</td></tr>
<tr><td colspan="3">Internal subprograms
or module subprograms</td></tr>
<tr><td colspan="3">END statement</td></tr>
</table>

© ISO/IEC 2010 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC 1539-1_2010

1 The syntax rules of clause 2.1 specify the statement order within program units and subprograms. These rules are illustrated in Table 2.1 and Table 2.2. Table 2.1 shows the ordering rules for statements and applies to all program units, subprograms, and interface bodies. Vertical lines delineate varieties of statements that may be interspersed and horizontal lines delineate varieties of statements that shall not be interspersed. Internal or module subprograms shall follow a CONTAINS statement. Between USE and CONTAINS statements in a subprogram, nonexecutable statements generally precede executable statements, although the ENTRY statement, FORMAT statement, and DATA statement may appear among the executable statements. Table 2.2 shows which statements are allowed in some kinds of scoping units.

Table 2.2: **Statements allowed in scoping units**

	Kind of scoping unit						
Statement type	Main program	Module or submodule	Block data	External subprogram	Module subprogram	Internal subprogram	Interface body
USE	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IMPORT	No	No	No	No	No	No	Yes
ENTRY	No	No	No	Yes	Yes	No	No
FORMAT	Yes	No	No	Yes	Yes	Yes	No
Misc. decl.s [1]	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DATA	Yes	Yes	Yes	Yes	Yes	Yes	No
Derived-type	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Interface	Yes	Yes	No	Yes	Yes	Yes	Yes
Executable	Yes	No	No	Yes	Yes	Yes	No
CONTAINS	Yes	Yes	No	Yes	Yes	No	No
Statement function	Yes	No	No	Yes	Yes	Yes	No

(1) Miscellaneous declarations are PARAMETER statements, IMPLICIT statements, type declaration statements, enumeration definitions, procedure declaration statements, and specification statements.

2.3.3 The END statement

1 Each program unit, module subprogram, and internal subprogram shall have exactly one END statement. The *end-program-stmt*, *end-function-stmt*, *end-subroutine-stmt*, and *end-mp-subprogram-stmt* statements are executable, and may be branch target statements (8.2). Executing an *end-program-stmt* initiates normal termination of the image. Executing an *end-function-stmt*, *end-subroutine-stmt*, or *end-mp-subprogram-stmt* is equivalent to executing a *return-stmt* with no *scalar-int-expr*.

2 The *end-module-stmt*, *end-submodule-stmt*, and *end-block-data-stmt* statements are nonexecutable.

2.3.4 Program execution

1 Execution of a program consists of the asynchronous execution of a fixed number (which may be one) of its images. Each image has its own execution state, floating-point status (14.7), and set of data objects, input/output units, and procedure pointers. The image index that identifies an image is an integer value in the range one to the number of images.

NOTE 2.5

Fortran control constructs (8.1, 8.2) control the progress of execution in each image. Image control statements (8.5.1) affect the relative progress of execution between images. Coarrays (2.4.7) provide a mechanism for accessing data on one image from another image.

© ISO/IEC 2010 – All rights reserved
AVA Documents
Document Sharing Hub
Full document details: ISO_IEC 1539-1_2010